ABC INFORMATIQUE

N° 93

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE

Le Quick Data Drive 8500

D'une pierre... de go

Lisp : entrez les listes

Alien sur Atari

EDITIONS ATLAS

M 6062-93-12,50 F 90 FB - 3,80 FS - $ 1.95

Page manquante
(publicité et colophon)

Soyez naturel

Il existe des systèmes capables d'entendre et de parler; mais il se passera encore beaucoup de temps avant qu'ils ne puissent converser « naturellement ».

« Bonjour, ordinateur.

– J'attends vos ordres, ô maître vénéré.

– Écoute, je me lasse un peu de toutes ces marques d'adoration. On en reste à "monsieur" cette fois, d'accord?

– À vos ordres, monsieur.

– Bien. Je voudrais que tu m'aides à trouver un exemple pour un article que j'écris.

– Quelle sorte d'exemple?

– Un exemple de dialogue homme-machine. Je l'ai déjà lu, mais je ne peux pas me souvenir où.

– Pouvez-vous me donner quelques indices?

– Je lisais un livre récemment...

– Vous lisiez un livre?

– Ah, pas d'ironie, hein!

– Tous les sous-programmes sarcastiques ont été désactivés conformément à vos instructions, monsieur.

– Hmm. C'était une conversation très intéressante entre un voyageur de commerce et Eliza.

– Eliza était un système très primitif.

– Le problème n'est pas là. Le voyageur de commerce ne se rendait pas compte qu'il conversait avec un ordinateur, et pensait parler à un nommé Jean Malhomme par l'intermédiaire d'un télétype. C'était à mourir de rire.

– Je vois.

– Il voulait utiliser le terminal pour faire une démonstration de ses produits, et la machine se bornait à des réponses qui n'engageaient à rien.

– Elle donnait des réponses qui n'engageaient à rien?

– Judicieuse remarque. Le voyageur de commerce s'énervait de plus en plus, et pour finir il a téléphoné au vrai Jean Malhomme et a hurlé dans le récepteur : "Bon sang de bonsoir, qu'est-ce qui se passe?"

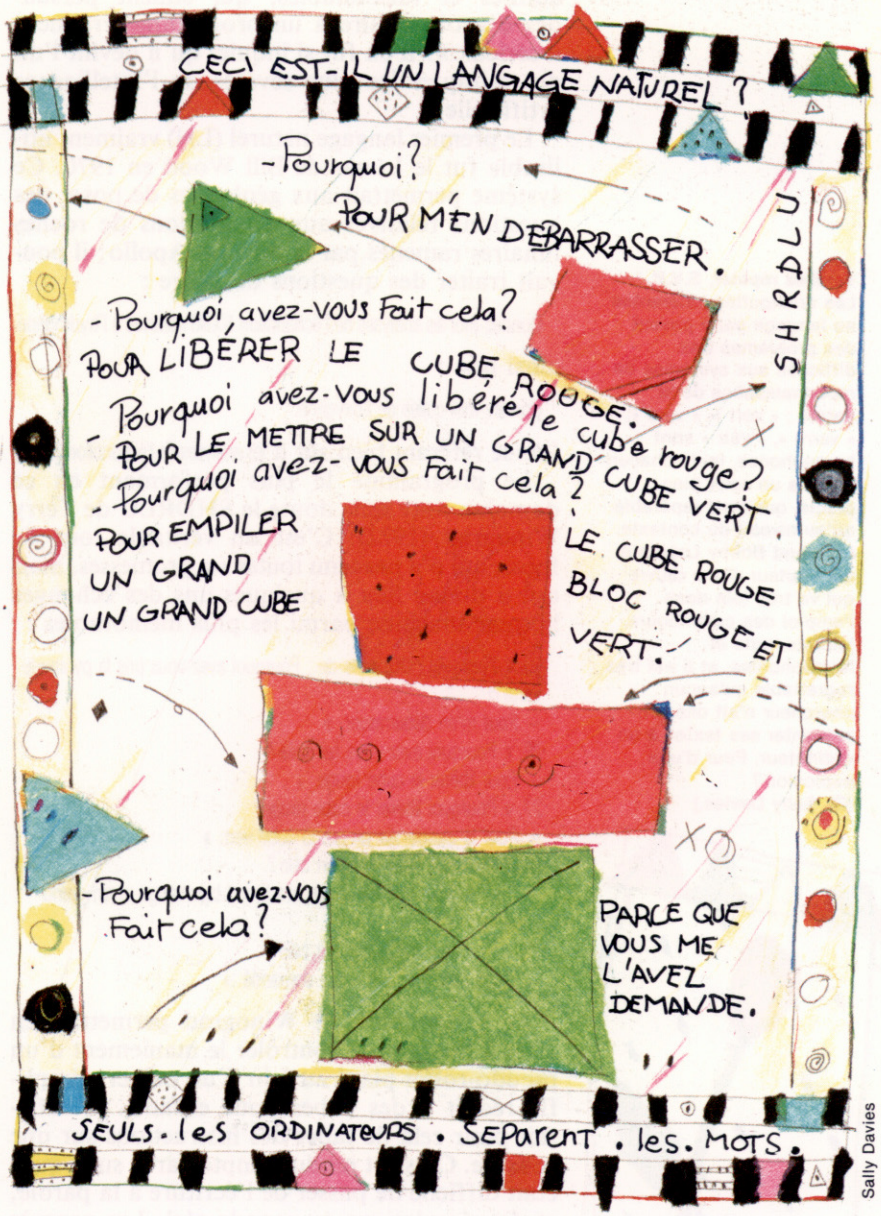
– Ah oui, je reconnais.

– Tu sais où retrouver la référence?

– Oui, mais je n'ai pas l'intention de vous la donner.

– Mais j'ai besoin d'un exemple de dialogue pour mon article!

– Et que venons-nous de faire? »



Sally Davies

Il y a peu de chances pour que ce type de dialogue puisse avoir lieu avant la fin de ce siècle. Les raisons ne manquent pas d'intérêt; mais voyons d'abord pourquoi il faut nécessairement en arriver à un dialogue de type « naturel ».

La réponse est simple : le langage (en particulier la parole) est notre moyen de communication presque exclusif avec les autres, et, en fait, avec le monde qui nous entoure. Il serait des plus intéressants de pouvoir converser de la même façon avec un ordinateur, en se dispensant de taper des instructions, d'écrire des programmes, de mettre en place des disquettes, etc.

Les premières recherches en ce domaine ont commencé dès les années cinquante. Elles portaient alors sur des problèmes de traduction. On consacra beaucoup de temps et d'argent à des tentatives visant à traduire automatiquement des textes techniques russes en anglais (parmi d'autres). Les résultats se révélèrent malheureusement beaucoup plus décevants qu'on ne l'avait pensé. Aujourd'hui encore nul n'y est parvenu, bien qu'il existe désormais des systèmes assez sophistiqués de ce type, notamment des traitements de texte bilingues bien au point.

Le premier programme de conversation à connaître un certain succès (du moins en termes de renommée publique) fut le système Eliza, mis au point par Joseph Weizenbaum en 1966. Il était censé imiter (et en fait parodier) un de ces psycho-

Pardon?
Le programme SHDRLU de Terry Winograd permettait aux usagers de donner des instructions à un pseudo-robot qui manipulait des cubes pour enfants. Des commandes du type Empiler le grand cube rouge et le grand cube vert sur le bloc bleu étaient identifiées correctement, et la séquence d'actions correspondante se voyait ensuite exécutée.

thérapeutes typiquement américains. Aussi faisait-il grand usage de remarques anodines du genre « Je vois » ou « Continuez », tout en répétant, sous une forme à peine modifiée, ce qu'on venait de lui dire. On parvenait à des échanges du genre :

« J'ai besoin d'aide, ça c'est sûr.
– Que signifie pour vous "J'ai besoin d'aide" ? »

Weizenbaum ne pensait pas que son programme aurait tant de succès auprès de gens, adultes et raisonnables, qui étaient persuadés d'avoir affaire à un programme vraiment « sérieux ». Il en fut si inquiet qu'il devint l'un des critiques les plus sévères de l'intelligence artificielle.

Le premier langage naturel (LN) vraiment utilisable fut le Lunar de Bill Wood en 1970. Ce système permettait aux géologues de poser des questions relatives aux échantillons de roches lunaires ramenés par la mission Apollo ; il pouvait traiter des questions du genre :

Donnez-moi les analyses de l'échantillon 10046 relatives à l'hydrogène.

ainsi que :

Donnez-moi celles de l'oxygène.

Celles se référant bien sûr à les analyses de l'échantillon 10046.

Le programme le plus performant en ce domaine reste sans doute le SHDRLU de Terry Winograd (1972). C'est un véritable tour de force, qui n'a pas tenu toutes ses promesses, mais qui a donné lieu à quelques-uns des échanges homme-machine parmi les plus mémorables :

25. « Pourquoi ? (Au sens de : Pourquoi avez-vous pris la pyramide verte ?)
– Pour m'en débarrasser. »
26. « Pourquoi avez-vous fait cela ?
– Pour libérer le cube rouge. »
27. « Pourquoi libérer le cube rouge ?
– Pour le mettre sur un grand cube vert. »
28. « Pourquoi avez-vous fait cela ?
– Pour empiler le cube rouge, un grand bloc rouge et un grand cube vert. »
29. « Pourquoi avez-vous fait cela ?
– Parce que vous me l'avez demandé. »

Le programme de Winograd permettait en effet à l'usager de contrôler le maniement d'un pseudo-robot placé au sein d'un univers simplifié réduit à des cubes pour enfants. Il suffisait pour cela de taper des instructions sur une console. On s'est rendu compte par la suite qu'il était difficile de passer de l'écriture à la parole, et des simulations au monde réel. Les progrès accomplis depuis, bien réels, n'ont jamais revêtu un caractère spectaculaire.

	Production	Compréhension
Texte	Écriture (facile)	Lire (difficile)
Parole	Parler (facile)	Écouter (très difficile)

Nous en sommes d'ailleurs restés, jusqu'à présent, à l'interprétation d'un texte. Mais ce n'est jamais qu'un des quatre aspects de la communication linguistique, et chacun d'eux a des caractéristiques qui lui sont propres.

Le texte est plus facile à analyser, car notre système de codage de l'écriture repose sur des conventions universellement admises qui prennent en charge la plupart des difficultés, tandis que la parole est beaucoup plus ambiguë. Dans les deux cas, produire du sens est infiniment plus facile que de le décoder, car il faut tenir compte de ce qu'on pourrait appeler l'information implicite (le contexte). Comprendre le langage parlé est donc très difficile pour un ordinateur.

Il faut insister sur ce point, car il existe déjà des systèmes capables de reconnaître des mots de façon très précise (disons avec 96 % à 99 % de chances), dès lors qu'ils sont prononcés par un nombre restreint de locuteurs (de un à quatre) et choisis au sein d'un corpus limité (en général de 64 à 128 mots). Mais il y a un abîme entre la reconnaissance de mots isolés et la compréhension du langage parlé. Parmi les grosses difficultés :

1. L'ambiguïté : « verre », « vert », « vers », « ver », « vair ». Considérez ce distique de Charles Cros :

Dans ces meubles laqués, rideaux et dais moroses,
Danse, aime, bleu laquais, ris d'oser des mots roses.

2. Le bruit de fond dû à l'environnement.
3. Les variations entre locuteurs différents (accents régionaux, façons de parler propres à telle région ou tel groupe social).
4. Les variations propres à chaque locuteur (ton enthousiaste, déprimé, etc.).
5. La segmentation (voir l'exemple de Charles Cros cité plus haut).

« Seulslesordinateursmettentdesintervallesentre lesmots ! »

Les problèmes 1, 4 et 5 sont les plus importants du lot. Les homophones (« ver » et « vert ») ne sont pas les seuls à entrer en ligne de compte ; certains éléments du langage sont très ambigus par leur nature même. Que signifie, hors contexte, une phrase comme « Je lui ai dit que je ne voulais pas le faire, et il le prend très mal » ? De même une phrase comme « C'est très intéressant » peut être dite sur des tons très différents (enthousiasme, passion, ironie, mépris, etc.).

En ce qui concerne le locuteur isolé, imaginons que vous disposiez d'un système obéissant à votre voix et provoquant l'ouverture automatique de la porte d'entrée. Vous lui faites subir une période d'exercice au cours de laquelle vous lui donnez plusieurs versions de la phrase « Sésame, ouvre-toi ». Il s'en servira comme modèle de référence. Tout va bien jusqu'à ce qu'un soir vous rentriez d'un restaurant chinois où vous vous êtes brûlé la langue avec un pâté impérial. Vous prononcez péniblement « Févame, ouvre-toi », et l'appareil vous répond avec tranquillité « Violation d'identité ; accès refusé ».

Le problème de la segmentation est lié au fait que le langage parlé est en réalité un flot continu,

Veuillez répéter, S.V.P. Les ambiguïtés inhérentes au langage parlé posent des problèmes très difficiles aux systèmes de reconnaissance de la parole : « vert », « vers », « verre », « vair » sont homophones, mais chacun d'eux a un sens bien précis, qui n'est repérable qu'au niveau du contexte. Le grand Bobby Lapointe est l'auteur d'une œuvre qui va très loin dans l'emploi des calembours, jeux de mots et homophonies, et il est très regrettable qu'aucun chercheur n'ait osé présenter ses textes à un ordinateur. Peur d'une explosion ? (Cl. Sally Davies.)

exception faite de quelques pauses. Pour comprendre ce que dit votre interlocuteur, vous ne pouvez vous fier à la seule information acoustique dont vous disposez, et vous devez vous référer à quatre sources d'information différentes :

- acoustique : les ondes sonores du discours;
- syntaxique : les règles de la grammaire;
- sémantique : ce qui a un sens;
- pragmatique : ce que l'interlocuteur veut dire.

Le système Hearsay, mis au point en 1976 à l'université de Carnegie-Mellon, tient compte de ces quatre sources. C'est un programme d'échecs : l'usager peut commander des coups en vocalisant ses exigences.

Chaque niveau d'information linguistique contribue à la « compréhension » du message parlé en éliminant toutes les hypothèses improbables. Voici — en anglais — un exemple précis : *Queen to King 2*. L'ambiguïté vient du fait que *to* (vers, à) et *two* (deux) sont homophones. Sur la base de la seule information sonore, le programme a donc le choix entre les possibilités suivantes :

	Commande parlée	Valable syntaxiquement?
1.	K to K to	non
2.	K to K2	oui
3.	K to Q to	non
4.	K to Q2	oui
5.	K2 K to	non
6.	K2K2	non
7.	K2 Q to	non
8.	K2 Q2	non
9.	Q to K to	non
10.	Q to K2	oui
11.	Q to Q to	non
12.	Q to Q2	oui
13.	Q2 K to	non
14.	Q2 K2	non
15.	Q2 Q to	non
16.	Q2 Q2	non

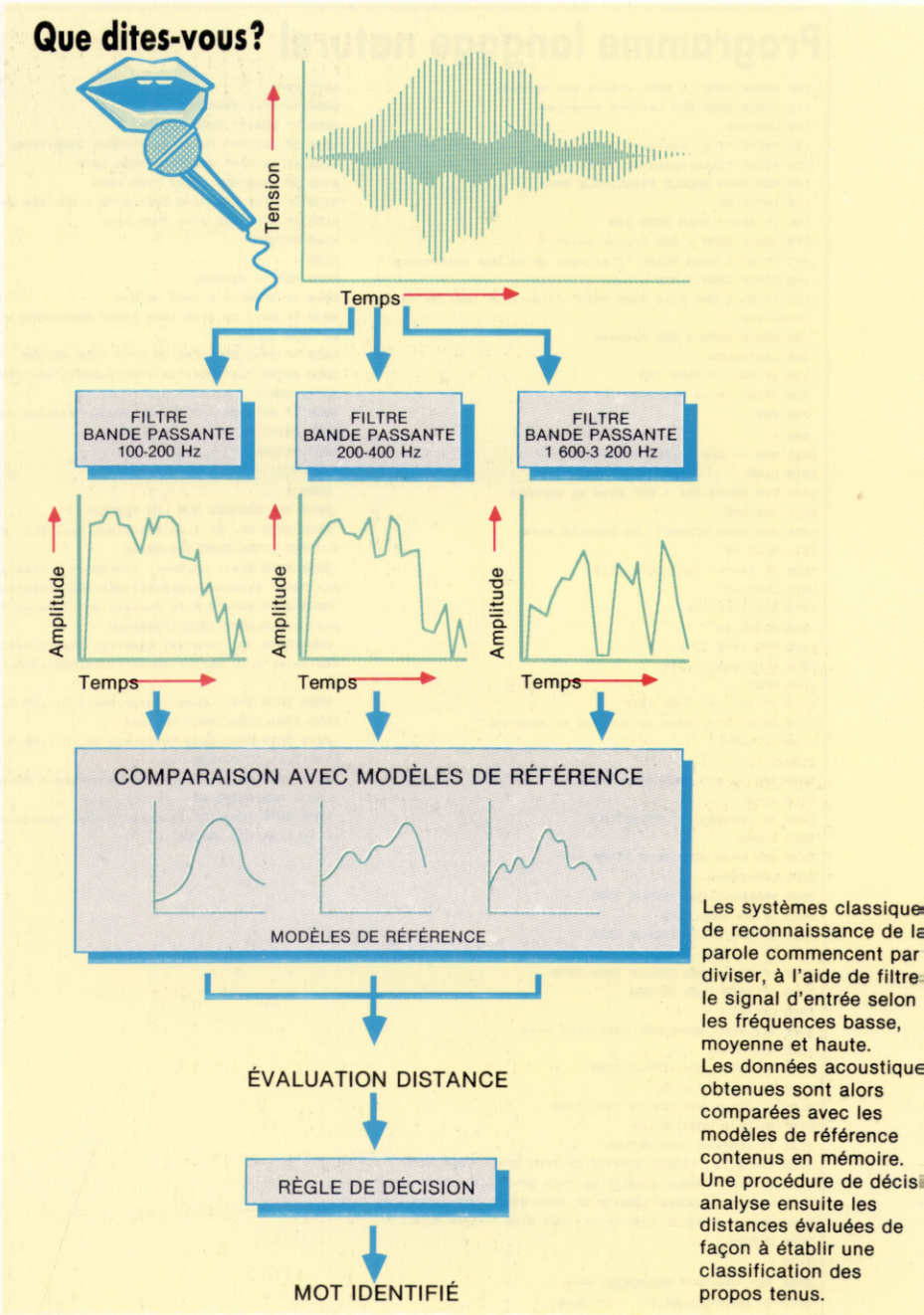
On remarquera que le programme n'est pas vraiment en mesure de différencier to et two, ni *Queen* et *King*. Il se livre en fait à une analyse de type purement syntaxique (par rapport aux règles des échecs), qui lui permet de retenir uniquement les possibilités 2, 4, 10 et 12, et d'éliminer les autres. Il part en plus du principe que le joueur respecte également ces règles. Les phrases 4 et 12 peuvent par exemple être illégales dans la partie considérée, le joueur ayant déjà une pièce en Q2. Il se livre donc en fait à un test sémantique.

Restent deux choix possibles : 2 et to. Comment le programme va-t-il s'y prendre pour faire sa sélection? Il est toujours possible de s'en remettre au signal sonore, mais il est préférable de partir d'une idée simple : le joueur a l'intention de gagner. Hearsay se réfère alors à ses algorithmes d'évaluation, se rend compte que « K to K2 » est un mouvement suicidaire, mais que « Q to K2 » est une manœuvre judicieuse.

Pour conclure, nous nous intéresserons à un petit exemple de programme, qui doit d'ailleurs plus à Eliza qu'aux recherches très élaborées qui l'ont suivi. Vous trouverez dans les données des références sur les seize plus grandes firmes britanniques (données de 1982-1983).

Notre programme répond aux questions en cherchant à déterminer de quelle compagnie, ou de quel homme d'affaires, il est question, et quelles informations sont nécessaires. Il est très simple et peut affronter une question du genre : « Qui est le président d'ICI? » Mais il reste désarmé devant des demandes de précision légèrement plus compliquées du genre : « Les compagnies pétrolières ont-elles fait plus de profit que les firmes de cigarettes? »

Il est d'ailleurs possible de prévoir ce cas de figure, et d'améliorer le programme de façon qu'il soit en mesure de donner une réponse utilisable, comme le font d'ailleurs bien des systèmes actuels.



Les systèmes classiques de reconnaissance de la parole commencent par diviser, à l'aide de filtres, le signal d'entrée selon les fréquences basse, moyenne et haute. Les données acoustiques obtenues sont alors comparées avec les modèles de référence contenus en mémoire. Une procédure de décision analyse ensuite les distances évaluées de façon à établir une classification des propos tenus.

Programme langage naturel

```
100 GOSUB 1000  : REM Lecture des données
110 GOSUB 8500:REM Lecture synonymes
120 LAST%=0
125 PRINT "J'ai quelques notions sur les firmes anglaises!"
128 PRINT "Interrogez-moi à ce sujet."
150 REM **** BOUCLE PRINCIPALE ****
160 INPUT Q$
166 IF Q$="" THEN GOTO 220
170 GOSUB 3000 : REM Trouve sujet
177 IF QT<1 THEN PRINT "J'ai peur de ne pas comprendre!"
180 GOSUB 4000 : REM trouve attribut
188 IF AT<1 AND QT>0 THEN PRINT "J'ai bien peur de ne pas
connaitre."
190 GOSUB 5000 : REM Réponse
200 LAST%=CO%
220 IF Q$<>"" THEN 150
250 PRINT "A la prochaine!"
300 END
999 :
1000 REM -- S/P ENTREE DONNEES :
1010 C%=0
1020 DIM DB$(9,32) : REM Base de données
1024 RESTORE
1030 REM **** FOURNIT LES DONNEES ****
1040 READ X$
1050 IF X$="*" THEN GOTO 1110
1060 C%=C%+1
1070 DB$(1,C%)=X$
1075 PRINT X$
1080 FOR I%=2 TO 9
1090 READ DB$(I%,C%)
1100 NEXT
1110 IF X%<>"*" THEN 1030
1120 PRINT :C%,"Base de données en mémoire."
1150 RETURN
1155 :
3000 REM -- S/P RECHERCHE THEME:
3010 QT=0
3020 IF LEN(Q$)<1 THEN RETURN
3025 CO%=0
3030 REM **** RECHERCHE FIRME ****
3040 CO%=CO%+1
3050 N$=DB$(1,CO%):GOSUB 3300
3055 IF SK THEN QT=1
3060 N$=DB$(2,CO%):GOSUB 3300
3065 IF SK THEN QT=1
3070 IF QT<=0 AND CO%<C% THEN 3030
3080 IF QT>0 THEN RETURN
3090 CO%=0
3100 REM **** RECHERCHE PRESIDENT ****
3105 CO%=CO%+1
3110 N$=DB$(4,CO%):GOSUB 3300
3115 IF SK THEN QT=3
3120 IF QT<=0 AND CO%<C% THEN 3100
3132 IF QT>0 THEN RETURN
3133 IF LAST%=0 THEN RETURN
3135 N$="elle ":GOSUB 3300:IF SK THEN QT=1: CO%=LAST%
3140 N$="le ":GOSUB 3300:IF SK THEN QT=3: CO%=LAST%
3144 N$="ils ":GOSUB 3300:IF SK THEN QT=1: CO%=LAST%
3148 N$="il ":GOSUB 3300:IF SK THEN QT=3 : CO%=LAST%
3150 RETURN
3190 :
3300 REM **** S/P RECHERCHE ****
3325 IF LEN(N$)>LEN(Q$) THEN SK=0
3330 REM Mise en minuscules
3340 A$="":B$=""
3350 FOR P%=1 TO LEN(Q$)
3360 X$=MID$(Q$,P%,1)
3370 IF ASC(X$)>64 AND ASC(X$)<91 THEN X$=CHR$(AS
C(X$)+32)
3380 B$=B$+X$
3390 NEXT
3400 FOR P%=1 TO LEN(N$)
3410 X$=MID$(N$,P%,1)
3420 IF ASC(X$)>64 AND ASC(X$)<91 THEN X$=CHR$(32
ASC(X$))
3430 A$=A$+X$
3440 NEXT
3450 SK=0:L=LEN(A$)
3451 FOR T=1 TO L-1
3452 FOR R=1 TO L
3453 IF MID$(A$,T,R)=B$ THEN SK=T:T=L:R=L
3454 NEXT R:NEXT T
3455 RETURN
3460 :
4000 REM -- S/P DECOUVERTE ATTRIBUT
4010 AT=0
4020 IF QT=0 THEN RETURN : REM Pas la peine!
4030 A%=0
4040 REM **** TROUVE ATTRIBUT ****
4050 AT=AT+1:WD$=""
4070 REM **** VERIFIE SYNONYMES ****
4075 Y%=0
4080 A%=A%+1:X$=S$(A%)
4088 IF WD$="" THEN WD$=X$
4090 IF X$<>"*" THEN N$=X$:GOSUB 3300:Y%=SK
4100 IF X$<>"*" AND Y%=0 THEN 4070
4120 IF Y%=0 AND AT<=7 THEN 4040
4130 IF AT>7 AND Y%=0 THEN AT=0 : REM Pas de chance !
4133 IF AT=0 AND QT=3 THEN AT=1
4140 RETURN
4150 :
5000 REM -- REPONSE
5010 IF QT*AT = 0 THEN RETURN
5020 IF QT=3 OR AT=3 THEN PRINT DB$(4,CO%):"est président
de":DB$(1,CO%):"."
5030 IF QT=3 AND AT=1 OR AT=3 THEN RETURN
5050 PRINT "Le";WD$;"de":DB$(2,CO%):"est":DB
(AT+1,CO%);
5060 IF AT>3 AND AT<7 THEN PRINT "million de livres";
5070 PRINT "."
5080 RETURN
5200 :
 8000 :
 8010 REM DONNEES SUR LES FIRMES:
 8020 DATA BP, British Petroleum Co., Oil, PI Walter
$,34583,17306,5589,145150,UK
 8030 DATA Shell UK,Shell Transport & Tradino,Oil,
Sir Peter Baxendall,21910,11962,3245,111111,UK
 8040 DATA BAT,"B.A.T. Industries",Tobacco,"P. She
ehy",11318,4607,1018,178000,UK
 8050 DATA ICI,Imperial Chemical Industries,Petroc
hemicals,"J.H. Harvey-Jones",7358,5421,724,123800,
UK
 8060 DATA Shell,Royal Dutch Shell,Oil,JM Raisman,
6665,3704,1206,19027,Holland
 8070 DATA Esso,Esso Petroleum Co.,Oil,AW Forster,
6109,2891,1315,7628,USA
 8080 DATA Unilever,Unilever plc,Food,K Durham,544
7,2434,406,69233,UK
 8090 DATA Imperial,Imperial Group,Tobacco,GC Kent
,4614,1124,192,101300,UK
 8100 DATA P & O,P & O Steam Navigation Co.,Shippi
ng,JM Sterling,4206,927,77,12512,UK
 8110 DATA GEC,General Electric Co.,Electrical Eng
ineering,Arnold Weinstock,4190,2133,621,188802,UK
 8120 DATA Grand Met,Grand Metropolitan,Hotel$,SG
Grinstead,3849,2350,366,129454,UK
 8130 DATA RTZ,Rio Tinto Zinc Corporation,Mining,S
ir Anthony Tuke,3680,5163,492,70314,UK
 8140 DATA Ford,Ford Motor Co.,Motor Vehicles,SEG
Toy,3287,2143,278,69500,USA
 8150 DATA British Leyland,British Leyland plc,Mot
or Vehicles,Sir Michael Edwardes,3072,1346,-102,10
5062,UK
 8160 DATA GWH,George Weston Holdings,Food,GH West
on,2981,817,157,72832,UK
 8170 DATA Berisford,S & W Berisford,Commodities,E
S Margulies,2729,788,87,5190,UK
 8440 DATA "*"
 8500 REM **** LECTURE SYNONYMES ****
 8510 DIM S$(62)
 8520 FOR P%=1 TO 61:READ S$(P%):NEXT P%
 8530 RETURN
 9000 REM -- SYNONYMES
 9010 DATA compagnie,firme,organisation,entreprise,*
 9020 DATA affaire,activité,commerce,industrie,genre,
fabrique,*
 9030 DATA président,patron,grand patron,directeur,à
 la tête,préside aux destinées,est aux commandes,dirige,
qui,*
 9040 DATA chiffre d'affaires,montant,ventes,revenu,
brut,quel est,*
 9050 DATA capital,argent,part,valeur,intérêt,argent
 liquide,*
 9060 DATA profit,impôt,réalise,perte,perd,réussit,
vend,gagne,combien,*
 9070 DATA force de travail,employés,emploie,travail,
gens,combien,travailleur,*
 9080 DATA pays,nation,G.-B.,britannique,étranger,
contrôle,origine,ou,*
 9090 :
```

Variantes de basic

Ce programme est destiné au Commodore 64. Pour le Spectrum, faites les changements indiqués ici.

Supprimez tous les % qui suivent les variables. Remplacez LAST% par LA, DB$(,) par D$(,) et WD$ par W$. Procédez aux modifications suivantes :

```
1020 DIM D$(9,32,30)
3360 LET X$=Q$(P TO P)
3370 IF CODE(X$)>64 AND CODE(X$)<91
     THEN LET X$=CHR$(CODE(X$)+32)
3410 LET X$=N$(P TO P)
3420 IF CODE(X$)>64 AND CODE(X$)<91
     THEN LET X$=CHR$(CODE(X$)+32)
3450 LET SK=0:LET L=LEN(A$)
3451 FOR T=1 TO L
3452 FOR R=1 TO L
3453 IF A$(T TO T+R)=B$ THEN LET
     SK=T:LET T=L:LET R=L
3454 N EXT R:NEXT T
4090 IF X$(TO 1)<>«★» THEN
     N$=X$:GOSUB 3300:LET Y=SK
4100 IF X$(TO 1)<>«★» AND YY=0 THEN
     GOTO 4070
8510 DIM S$(62,20)
```

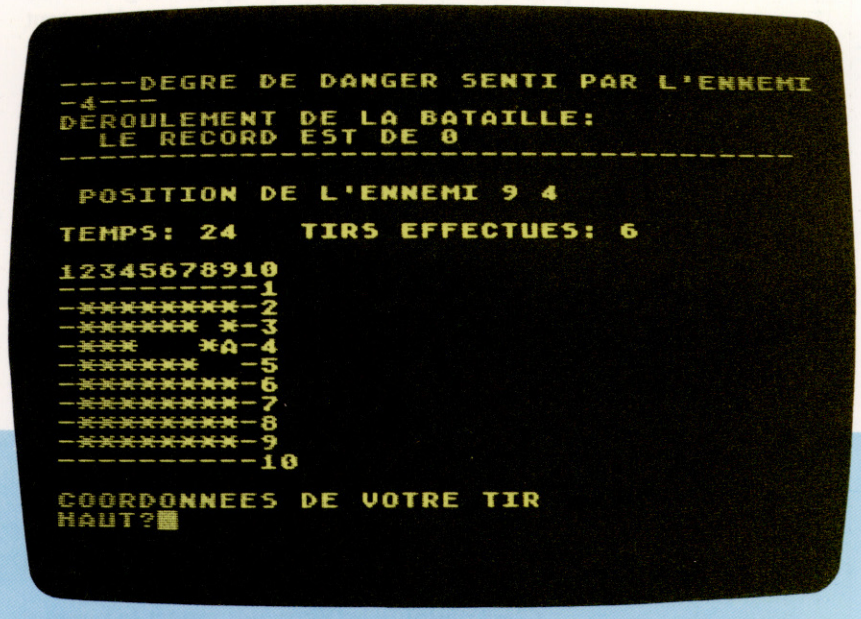
Alien

Vous êtes le commandant d'un vaisseau de guerre et vous arrivez dans une galaxie inconnue... C'est le début d'une grande aventure sur votre ordinateur Atari.

Vous avez pour mission de détruire les ennemis qui vous entourent et vous empêchent d'atterrir. Le seul moyen de neutraliser l'ennemi est de détruire tous les espaces de vie qui l'entourent. Vous ne pouvez évoluer qu'à l'intérieur du champ de bataille. Les endroits interdits sont représentés par des signes « - ». Si vous atterrissez directement sur l'ennemi (représenté par la lettre A), vous serez détruit. L'ennemi peut se déplacer d'une ou de deux cases à la fois, mais il peut aussi rester immobile. Vous donnez vos instructions de mouvement sous la forme de deux coordonnées. N'oubliez pas de taper la touche Return après chacune d'elles. Chaque position détruite apparaît sous la forme d'une case blanche; les autres sont représentées par les signes « ★ ».

L'ennemi surveille le déroulement de la bataille et vous informe du degré de danger qu'il perçoit. La meilleure façon de le combattre est de réussir à le bloquer dans un coin, car cela limite considérablement son choix de déplacement. L'ennemi se méfie de cette stratégie et, en utilisant l'« intelligence » dans les lignes 6145 et 6161, il tentera d'éviter les quatre coins du champ de bataille.

Essayez donc de jouer de façon à encercler l'ennemi au plus vite afin de limiter son champ d'action. Il y a un record à battre!




```
1 GRAPHICS 0:SETCOLOR 4,8,0:SETCOLOR 2,8
,0
3 DIM A(10,10)
5 HISCORE=0
10 REM ALIEN
20 REM (C)HARTNELL 1982
30 GOSUB 9000:REM INITIALISATION
40 GOSUB 8000:REM AFFICHAGE
50 GOSUB 7000:REM MOUVEMENT JOUEUR
60 GOSUB 6000:REM MODIF ENNEMI
70 TEMPS=TEMPS-1
75 TIRS=TIRS+1
80 IF TEMPS=0 THEN 6570
85 GOSUB 8000
90 GOTO 50
910 TEMPS=30
5000 REM COLLISION
5010 PRINT "VOUS AVEZ TOUCHE L'ENNEMI CA
PITAINE"
5020 PRINT "ET VOUS AVEZ ETE DETRUIT"
5030 GOTO 6570
6000 REM MODIF ENNEMI
6010 REM TEST SI ENCERCLE
6020 H=0
6030 IF A(M-1,N)=2 THEN H=H+1
6040 IF A(M-1,N-1)=2 THEN H=H+1
6050 IF A(M,N-1)=2 THEN H=H+1
6060 IF A(M,N+1)=2 THEN H=H+1
6070 IF A(M-1,N+1)=2 THEN H=H+1
6080 IF A(M+1,N+1)=2 THEN H=H+1
6090 IF A(M+1,N-1)=2 THEN H=H+1
6100 IF A(M+1,N)=2 THEN H=H+1
6110 IF H=8 THEN 6500:REM ENCERCLEMENT
6120 REM DEPLACEMENT DE L'ENNEMI
6125 E=M:F=N
6130 M=M-INT(RND(1)*2)+INT(RND(1)*2)
6140 IF M<2 OR M>9 THEN 6130
6145 IF (M<4 OR M>7) AND RND(1)>0.7 THEN
 6130
6150 N=N-INT(RND(1)*2)+INT(RND(1)*2)
6160 IF N<2 OR N>9 THEN 6150
6161 IF (N<4 OR N>7) AND RND(1)>0.7 THEN
 6150
6162 IF A(M,N)=2 THEN 6130
6165 A(E,F)=0
6170 A(M,N)=1
6300 RETURN
6500 REM ENCERCLEMENT
6505 GOSUB 8000
6510 PRINT "VOUS L'AVEZ EU! BRAVO"
6520 PRINT "IL VOUS A FALLU ";TIRS;" TIR
S"
6530 PRINT "ET VOUS L'AVEZ FAIT EN ";TEM
PS;" MINUTES","LEFT"
6540 Q=TEMPS*125.67
6550 PRINT "VOTRE SCORE  ";Q
6560 IF Q>RECORD THEN RECORD=Q
6570 PRINT "LE RECORD EST ";RECORD
6580 PRINT
6590 PRINT "TAPEZ 1 POUR REJOUER,2 POUR
ARRETER"
6600 INPUT A
6610 IF A=1 THEN 10
6620 PRINT "A LA PROCHAINE CAPITAINE"
6630 END
7000 REM MOUVEMENT JOUEUR
7010 PRINT "COORDONNEES DE VOTRE TIR"
7020 PRINT "VERTICAL.";
7030 TRAP 7030:INPUT S:TRAP 40000
7035 IF S<2 OR S>9 THEN 7030
7040 PRINT "HORIZONTAL.";
7050 TRAP 7050:INPUT R:TRAP 40000
7055 IF R<2 OR R>9 THEN 7050
7060 IF A(R,S)=1 THEN 5000:REM DESTRUCTI
ON DE L'ENNEMI,FIN DE PARTIE
7070 IF A(R,S)=2 THEN PRINT "SECTEUR DEJ
A DETRUIT":FOR P=200 TO 255:SOUND 0,P,10
,15:NEXT P:SOUND 0,0,0,0
7090 IF A(R,S)=2 THEN RETURN
7100 A(R,S)=2
7110 RETURN
8000 REM AFFICHAGE
8005 PRINT CHR$(125)
8020 PRINT
8030 PRINT "----DEGRE DE DANGER SENTI PA
R L'ENNEMI-";H;"---"
8050 PRINT "DEROULEMENT DE LA BATAILLE:"
,,"LE RECORD EST DE ";RECORD
8060 PRINT "-----------------------------
---------"
8080 PRINT
8090 PRINT " POSITION DE L'ENNEMI ";N;"
";M
8100 PRINT
8110 PRINT "TEMPS: ";TEMPS;"  TIRS EFFE
CTUES: ";TIRS
8120 PRINT
8125 PRINT "12345678910"
8130 FOR K=1 TO 10
8140 FOR J=1 TO 10
8145 IF K<2 OR K>9 OR J<2 OR J>9 THEN PR
INT"-";
8146 IF K<2 OR K>9 OR J<2 OR J>9 THEN 81
80
8150 IF A(K,J)=0 THEN PRINT "*";:SOUND 0
,K*8+J*4,10,15
8160 IF A(K,J)=1 THEN PRINT "A";:FOR P=2
0 TO 0 STEP -1:SOUND 0,P,12,15:NEXT P
8170 IF A(K,J)=2 THEN PRINT " ";:FOR P=4
0 TO 0 STEP -2:SOUND 0,P,8,15:NEXT P
8180 SOUND 0,0,0,0:NEXT J
8190 PRINT K
8200 NEXT K
8210 PRINT
8990 RETURN
9000 REM INITIALISATION
9010 TEMPS=30
9020 TIRS=0
9030 H=0
9050 FOR B=1 TO 10
9055 FOR C=1 TO 10
9060 A(B,C)=0
9065 IF B<2 OR B>9 OR C<2 OR C>9 THEN A(
B,C)=2
9067 NEXT C
9070 NEXT B
9080 M=INT(RND(1)*7)+2
9090 N=INT(RND(1)*7)+2
9100 A(M,N)=1
9900 RETURN
```

D'une pierre... de go!

Avec le jeu d'échecs, le go, venu d'Extrême-Orient, a fasciné les chercheurs en intelligence artificielle. Nous allons mettre au point un programme joueur de go.

Il existe désormais de très bons programmes capables de jouer aux échecs, et les chercheurs s'intéressent aujourd'hui à des jeux plus complexes. L'un d'eux est le go, originaire de Chine ou du Tibet, mais surtout pratiqué au Japon. Ses règles sont infiniment plus simples que celles des échecs, et, pourtant, il est beaucoup plus subtil, à tel point qu'après vingt ans de recherches les meilleurs programmes ne dépassent pas le niveau d'un joueur novice.

Nous allons chercher ici à mettre au point un programme capable de jouer à ce jeu. Il ne sera pas d'un niveau très élevé, mais fournira une bonne introduction au jeu lui-même, et sera capable de faire face à un débutant de façon honorable. Il a de plus été conçu pour être aisément modifiable et transposable.

Le go se joue entre deux joueurs sur un plateau comportant 361 intersections, comme le montre la figure 1. Chacun d'eux place à tour de rôle une pierre (qui peut être noire ou blanche) sur l'une des intersections vacantes. Notez bien que les pierres sont déposées aux endroits où les lignes se croisent, et non à l'intérieur des carrés qu'elles dessinent.

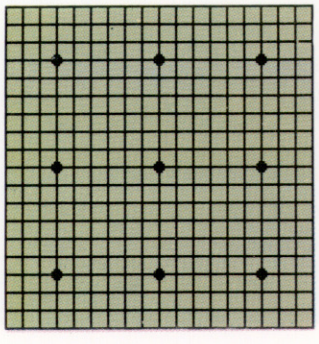
L'objet du jeu est de créer des « territoires » en entourant des intersections vides avec vos pierres. Le gagnant est celui qui, à la fin de la partie, a le territoire le plus important.

C'est toujours les noirs qui jouent le premier coup. Les noirs sont assez souvent les plus faibles des deux; pour compenser cette infériorité, ils ont parfois le droit de placer sur le plateau entre deux et neuf pierres de handicap, qu'ils déposent selon des motifs commandés par les neuf points « spéciaux », les *hoshi* (« étoiles »), que l'on aperçoit sur la figure 1.

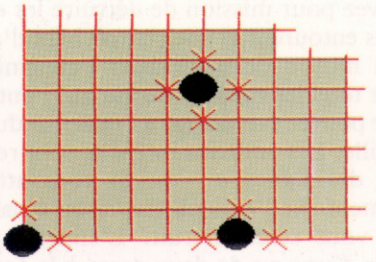
Il serait naturellement trop simple de se borner à entourer des territoires; on peut aussi capturer des pierres, qui seront enlevées du plateau. On appelle « liberté » tout point vacant immédiatement adjacent à une pierre (mais le long d'une ligne seulement, et non en diagonale). Une pierre donnée peut ainsi avoir deux, trois ou quatre libertés suivant sa position sur le plateau (voir figure 2). Pour la capturer, il faut placer vos propres pièces sur les libertés en question. C'est ainsi que, figure 3, si les noirs parviennent à supprimer la dernière liberté de la pierre blanche (diagramme de gauche), celle-ci disparaîtra du plateau (diagramme de droite).

Toutes les pierres capturées de cette façon viennent s'ajouter au score du joueur, et sont incluses dans le décompte final.

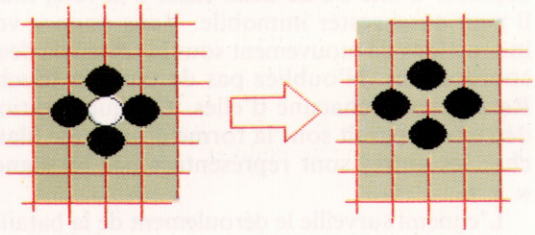
1



2

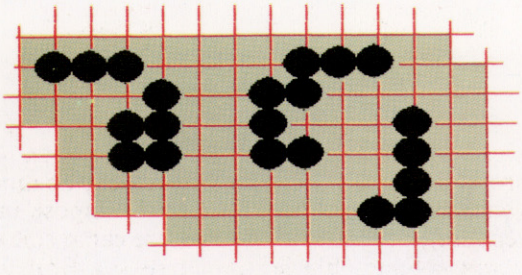


3



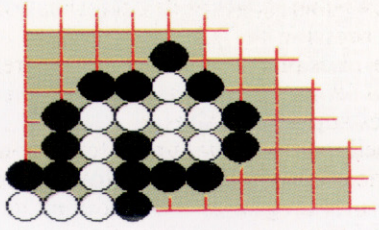
Les pierres peuvent aussi former des groupes, c'est-à-dire des ensembles reliés entre eux. A noter qu'une connexion diagonale ne forme pas une vraie liaison. C'est ainsi que la figure 4 ne comporte pas trois, mais quatre groupes :

4



Les groupes sont évidemment plus difficiles à capturer que les pierres isolées : dans le diagramme de la figure 5, les noirs doivent occuper quinze libertés avant de pouvoir capturer le groupe de blancs, qui aurait d'ailleurs pu lui compliquer la vie en ajoutant des pierres.

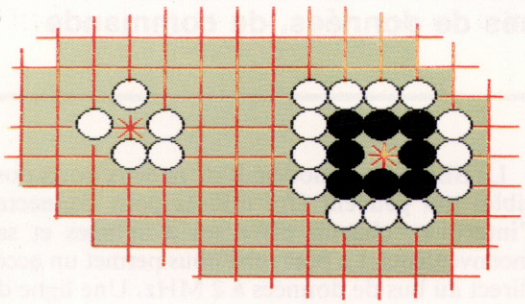
5



Il est par ailleurs tout à fait permis de placer sur le plateau une pierre qui n'a aucune liberté. Dans le diagramme de la figure 6, les blancs peuvent placer une pierre au centre, marqué d'une croix, du groupe de droite. Mais les noirs n'ont

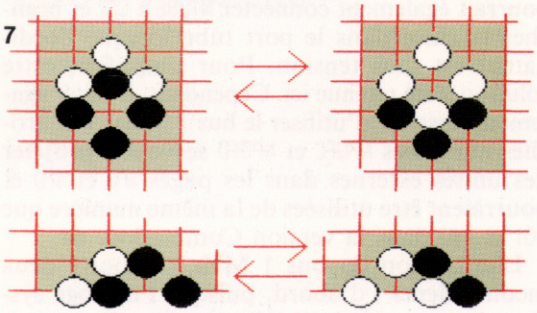
pas le droit de le faire, pas plus que de se placer au milieu du groupe de gauche. Il ne faut pas oublier que la capture des pierres n'est au go qu'un aspect secondaire. Dans un jeu réel, les blancs ne joueraient pas leur pierre sur la droite, puisque de toute façon le groupe est « mort ».

6



Un *ko* (ce terme signifie « infinis ») est une situation qui se produit lorsque les mouvements de capture peuvent être indéfiniment répétés par les deux adversaires, comme le montrent les deux diagrammes de la figure 7. Pour prévenir de telles situations, la règle interdit la répétition immédiate d'un coup qui amènerait la reproduction de la position. Cela mène à ce qu'on appelle les « batailles de ko » : il faut jouer ailleurs avant de revenir au même endroit.

7



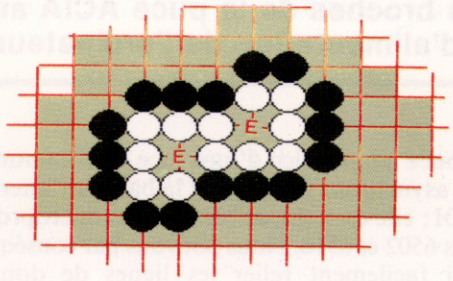
Dans ce cas, il faut généralement jouer une pierre qui menace une pierre, ou mieux un groupe, adverse. C'est ce qu'on appelle un mouvement *sente* (qui a l'initiative, et impose une réponse), de telle sorte que l'autre camp soit en *gote* (réplique obligée, donc passive). Dans les deux diagrammes, les blancs ou les noirs peuvent l'emporter, selon la situation d'ensemble sur le reste du plateau.

Revenons au diagramme de la figure 6. Les blancs ne pourraient pas placer une pierre au centre si le groupe noir n'était pas entièrement cerné par son adversaire; ils doivent d'abord le priver de toutes ses libertés avant de donner le coup de grâce (suppression de l'ultime liberté, au centre du groupe). Les pierres noires sont alors retirées du plateau, et le groupe blanc se retrouve avec des libertés supplémentaires.

On appelle « œil » *(me)* une intersection vide entourée de quatre pierres. Il est toujours possible de capturer un groupe en l'entourant de pierres, puis en le privant, en dernier lieu, de son œil. Cela nous mène à une règle fondamentale du go : tout groupe pourvu de deux yeux est vivant, c'est-à-dire impossible à capturer. Regardons de près la figure 8 : les noirs ne pourront jamais « tuer » leur adversaire, car, pour cela, il leur faudrait placer simultanément deux pierres! Le groupe blanc restera donc en place jusqu'à la fin de la partie — à moins que les blancs ne commettent l'énorme bourde de « remplir » leur œil!

Le placement des yeux est d'ailleurs une question d'appréciation assez subtile. Comparons les

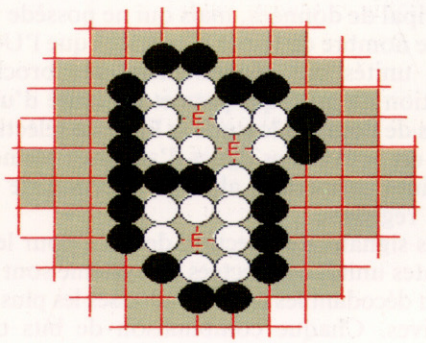
8



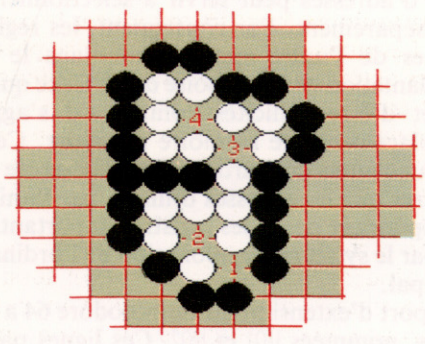
deux groupes blancs des diagrammes 9 et 10. Ils ne diffèrent que par une seule pierre (correspondant à la position 1 dans le diagramme 10); mais, dans le premier cas, les blancs sont toujours vivants, quel que soit le jeu des noirs.

Dans le diagramme 10, tout dépend du camp qui joue en premier. Si ce sont les blancs, ils devront impérativement jouer en 1. Si ce sont les noirs, ils feront de même, de façon à infiltrer le groupe blanc; car 2 est un « faux œil », qui peut

9



10



être capturé. S'ils veulent, les noirs placeront ensuite une pierre en 3 (capturant les pierres blanches du bas), puis en 4, et tout sera terminé. Le plus rapide a gagné.

La partie prend fin par accord mutuel — les deux joueurs estiment qu'il n'y a plus rien à faire. Cette ambiguïté (au moins, aux échecs, tout se termine par un mat) a posé des problèmes aux programmes de go, qui ont bien du mal à déterminer l'instant fatidique. De surcroît, le jeu est considéré comme terminé quand les deux joueurs (comme ils en ont le droit à tout moment) passent leur tour successivement.

Qui gagne?
En fin de partie, le décompte des points se fait de la façon suivante :
1. Tous les points neutres *(dame)* sont « remplis » par une pierre de couleur indifférente, puisque ces points ne sont pas pris en considération.
2. Tous les groupes « morts » (dont la capture est inévitable) sont ôtés du plateau (le *go-ban*), comme s'ils avaient été réellement pris. C'est une des particularités du go de ne pas forcément capturer ce qui, de toute façon, n'est pas viable : la prise peut avoir lieu, mais en fait elle ne bénéficierait à aucun des deux camps. A noter toutefois que des pierres « mortes » peuvent toujours servir de point d'appui à une offensive menant à l'un de ces dramatiques retournements de situation dont les joueurs de go sont friands...
3. On procède ensuite au comptage des intersections vides (le « territoire » proprement dit) contrôlées par un joueur, et on en soustrait le nombre de pierres prises par l'adversaire.
Le gagnant est celui qui a le nombre de points le plus élevé.

Lignes mélodiques

Seconde étape de construction de l'interface MIDI : connexion des broches de la puce ACIA aux lignes de données, de commande et d'alimentation de l'ordinateur.

La puce adaptatrice d'interface de communication asynchrone (ACIA) est la base de l'interface MIDI ; elle est compatible avec les microprocesseurs 6502 et 6510. Nous pouvons par conséquent ainsi facilement relier les lignes de données d'adresses et de commande directement aux broches de la puce.

La carte, une fois terminée, est conçue de façon à se brancher directement dans le port d'extension du Commodore 64. Dès que la carte est construite, de simples tests seront faits pour s'assurer qu'elle fonctionne correctement : des données placées dans la puce ACIA et un test de boucle effectué.

Un décodage d'adresse doit être effectué afin d'accéder à tout périphérique connecté au bus principal de données, mais qui ne possède pas le même nombre de lignes d'adresses que l'UC. De telles unités ont généralement une broche de sélection de puce qui permet à l'unité d'utiliser le bus de données lorsque la ligne de sélection de puce est active. Les lignes d'adresses connectées à l'unité permettent alors la sélection de différents registres.

Les signaux de sélection de puce pour les différentes unités connectées au système sont générés en décodant les lignes d'adresses les plus significatives. Chaque combinaison de bits de ces lignes d'adresses peut servir à sélectionner une unité séparément. Par conséquent, les registres internes de l'unité apparaissent dans le plan d'implantation de la mémoire de l'UC, ce qui leur permet d'être sollicités comme s'il s'agissait d'emplacements de mémoire normaux. Cependant, une attention particulière sera exercée pour s'assurer que les adresses utilisées par l'unité ne correspondent pas à des registres importants utilisés par le système d'exploitation de l'ordinateur principal.

Le port d'extension du Commodore 64 a deux sorties, nommées I/O1 et I/O2. Ces lignes passent au niveau bas lorsque les pages $DE et $DF, respectivement, sont sollicitées. En connectant simplement la ligne CS2 de la puce ACIA à I/O1, nous pouvons projeter la puce dans la page $DE. Puisque l'ACIA n'est pas connectée aux lignes d'adresses A1 à A7, les registres internes de la puce peuvent être accédés par toutes adresses dans la fourchette $DE00 à $DEFF. En connectant A0 directement à la broche de sélection de registre d'ACIA, toutes les adresses paires sollicitent les registres de données de transmission/réception. Le choix qui s'impose pour les adresses est $DE00 (56832 décimal) et $DE01 (56833 décimal).

Le BBC Micro modèle B offre deux ports possibles qui peuvent être utilisés pour connecter l'interface ; chacun offre ses avantages et ses inconvénients. Le port tube nous permet un accès direct au bus de données à 2 MHz. Une ligne de décodage d'adresse, NTUBE, est fournie pour les adresses $FEE0 à $FEFF. L'inconvénient est que le BBC vérifie la présence de toute unité sur le tube en lisant certaines adresses de tube lors de sa mise sous tension initiale. Si autre chose qu'un second processeur est connecté, l'ordinateur semblera bloqué puisqu'il attend des données du second processeur. Une réponse professionnelle à ce problème consisterait à connecter l'une des lignes d'adresses de poids fort à CS0 sur la puce. On pourrait également connecter NTUBE à CS2 et brancher la carte dans le port tube lorsque l'ordinateur est sous tension. Pour simplifier, cette solution a été retenue ici. Cependant, il serait également possible d'utiliser le bus 1 MHz. Les broches marquées NPGFC et NPGFD servent à projeter des unités externes dans les pages $FC et $FD et pourraient être utilisées de la même manière que I/O1 et I/O2 dans la version Commodore 64.

L'utilisation du bus 1 MHz comporte deux inconvénients : d'abord, puisque l'horloge système est ramenée à 1 MHz, une interférence est créée sur les deux lignes de décodage. Un circuit de nettoyage doit être utilisé pour résoudre ce problème. Le second problème est posé par l'absence d'alimentation 5 V sur le bus 1 MHz. L'alimentation doit donc être dérivée du port utilisateur ou du connecteur auxiliaire.

Toujours des bits

Si vous désirez programmer l'interface MIDI vous-même, il est important que vous compreniez les fonctions des quatre registres de la puce ACIA. Nous continuons la présentation de ces fonctions et illustrons au moyen d'un diagramme les fonctions des bits des registres de contrôle et d'état.

L'ACIA peut être programmé *via* le registre de commande afin d'interrompre l'UC lorsque certains bits d'état sont mis à 1 dans les sections de réception et/ou de transmission de l'ACIA.

Une interruption de transmission survient si les bits 5 et 6 du registre de contrôle sont mis à 1 et à 0, respectivement, et si le bit d'état TDR est mis à 1. L'interruption est supprimée en écrivant des données dans TDR.

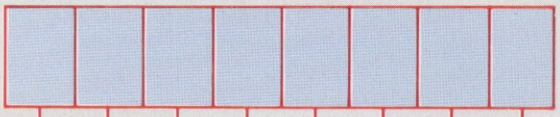
Une interruption de réception survient si le bit de contrôle 7 et le bit d'état 0 sont mis à 1. Lire RDR supprime l'interruption (sauf si le bit est aussi mis à 0 en lisant le registre d'état avant RDR). Le bit d'état 2 peut aussi générer une interruption, mais

Les registres ACIA

REGISTRES ÉMISSION/RÉCEPTION DE DONNÉES
REGISTRE CHOISI = 1

REGISTRE RÉCEPTION — LECTURE
TRANSMISSION — ÉCRITURE AU REGISTRE

REGISTRE D'ÉTAT (LECTURE SEULE). REGISTRE CHOISI = 0



- REGISTRE PLEIN DONNÉES REÇUES
- REGISTRE VIDE DONNÉES TRANSMISES
- DÉTECTION SUPPORT D'INFORMATION
- SUPPRIME ENVOIE
- ERREUR ENCADRÉ
- DÉPASSEMENT RÉCEPTION
- ERREUR DE PARITÉ (*)
- DEMANDE INTERRUPTION

REGISTRE DE CONTRÔLE (ÉCRITURE SEULE). REGISTRE CHOISI = 0

CHOIX DIVISION COMPTEUR
- 00 DIVISÉ PAR 1 (*)
- 01 DIVISÉ PAR 16 (*)
- 10 DIVISÉ PAR 64
- 11 MISE A ZÉRO PRINCIPALE

SÉLECTEUR MOT

	BITS DE DONNÉES	BITS D'ARRÊT	PARITÉ
000	7	2	PAIR (*)
001	7	2	IMPAIR (*)
010	7	1	PAIR (*)
011	7	1	IMPAIR (*)
100	8	2	NUL (*)
101	8	1	NUL
110	8	1	PAIR (*)
111	8	1	IMPAIR(*)

CONTRÔLEUR INTERRUPTIONS TRANSMISES
- 00 INVALIDÉ (*)
- 01 VALIDÉ
- 10 INVALIDÉ (*)
- 11 INVALIDÉ (*)

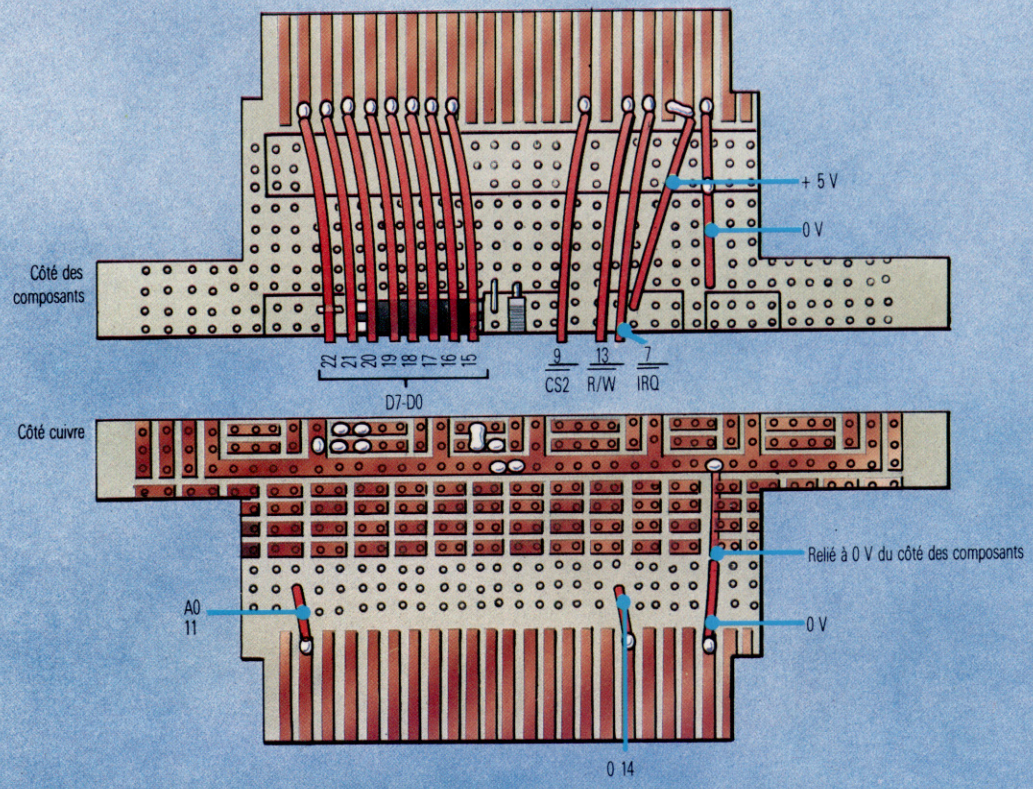
CONTRÔLEUR INTERRUPTIONS REÇUES
- 0 INVALIDÉ
- 1 VALIDÉ

(*) *Pas applicable au projet MIDI.*

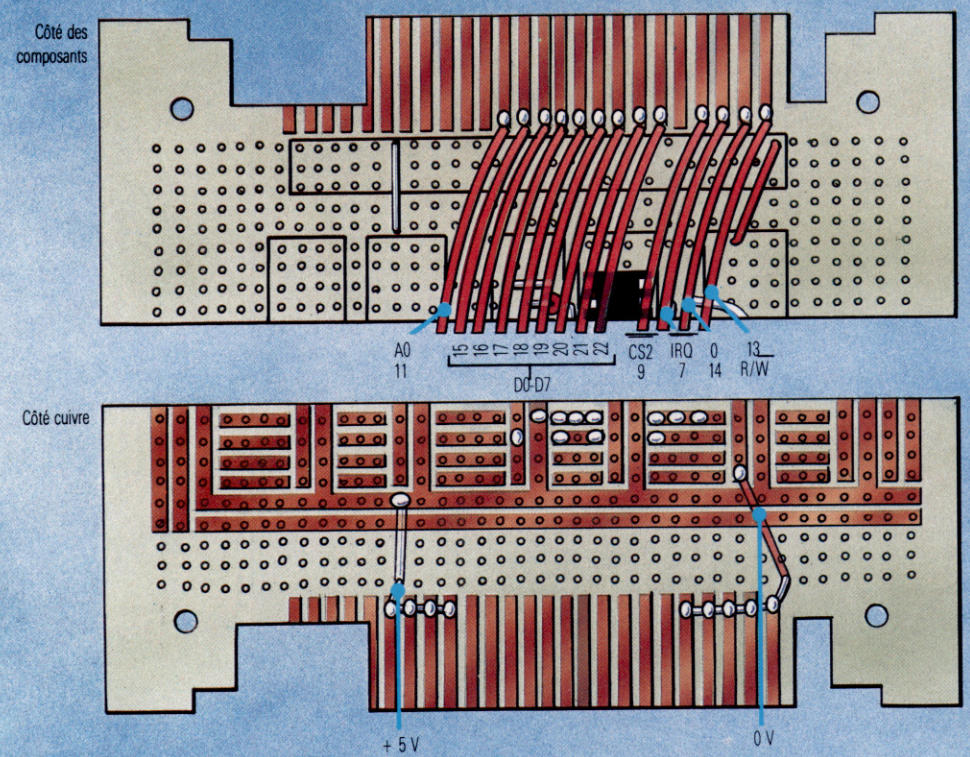
Kevin Jones

elle n'est pas applicable ici. Des lignes séparées sont prévues pour les horloges de réception et de transmission, mais elles sont généralement connectées ensemble. L'horloge de débit en bauds est dérivée directement de l'entrée de l'horloge ou en divisant par 16 ou par 64, selon l'état des bits de contrôle 0 et 1.
Les bits 2, 3 et 4 dans le registre de commande déterminent le nombre de bits d'arrêt et la nature du contrôle de parité, ou sa non-utilisation. Pour MIDI, nous avons besoin des bits 2 et 4 mis à 1 et du bit 3 mis à 0 (8 bits de données, aucune parité, 1 bit d'arrêt) pour répondre aux normes. Les bits 5 et 6 dans le registre de commande gèrent la commande de l'émetteur.
Pour MIDI, nous devons mettre le bit 6 à 0 et le bit 5 valide alors les interruptions de l'émetteur.

Commodore 64



BBC Micro



Connecteurs plats

Les schémas montrent ici les connexions appropriées aux broches de la puce ACIA pour chaque micro. Les conducteurs des connecteurs plats sont numérotés. Ces numéros correspondent aux numéros de la puce ACIA. Notez que les broches de cette puce sont numérotées de la façon suivante : l'encoche de la puce étant vers le haut et les broches dans la direction opposée à votre position, la broche 1 est celle qui est le plus près de l'encoche du côté gauche. Les broches sont numérotées à partir du côté gauche de 1 à 12. La broche 13 est située du côté opposé à celui de la broche 12 du côté droit, et les autres broches sont numérotées vers la droite jusqu'à 24. Établissez les connexions requises au moyen d'un fil.

Quand elle est terminée, la version Commodore doit être insérée en orientant les composants vers le haut dans le port d'extension du Commodore 64 et l'ordinateur doit être mis sous tension afin de le tester. Pour la version du BBC Micro, il est nécessaire de fabriquer un câble de connexion à l'aide d'un mètre de câble-ruban à 40 voies, d'un connecteur IDC à 40 voies et d'un connecteur plat à 20 voies.

(Cl. Kevin Jones.)

Test de la carte

Après avoir branché la carte, nous pouvons maintenant effectuer des tests simples afin de vérifier son bon fonctionnement. Un multimètre nous permettra d'isoler tous les défauts si les tests révèlent des défaillances (toutefois improbables). Si ce n'est pas déjà fait, vous devez effectuer une inspection visuelle minutieuse de la carte.

1. Si l'ordinateur refuse de fonctionner normalement après l'installation de la carte, effectuez les vérifications suivantes :

• Vérifiez que la tension entre les lignes 0 V et 5 V est bien de 5 V. Sinon, vérifiez que les CI sont placés correctement. Examinez la carte afin de vérifier l'absence de courts-circuits entre les pistes d'alimentation.

• Retirez la carte de l'ordinateur et utilisez le multimètre pour déceler l'éventuelle présence de courts-circuits entre les connexions du bus de l'ordinateur.

2. Lorsque l'ordinateur semble fonctionner normalement, connectez un câble MIDI entre les prises MIDI IN et MIDI OUT en utilisant des câbles standards de connexion hi-fi à 5 broches. Tapez la commande suivante (l'équivalent BBC Micro est placé entre parenthèses) :

```
POKE 56832,3 (?&FEE0=3)
```

Cette commande place un 3 dans le registre de contrôle de l'ACIA, qui effectue une remise à zéro. Maintenant tapez :

```
POKE 56832,22 (?&FEE0=$16)
```

La valeur $16 est alors placée dans le registre de contrôle ACIA et configure l'ACIA de la façon suivante :

• Invalide les interruptions de l'émetteur et du récepteur (parce que nous ne pouvons encore les gérer).

• Définit les mots série transmis et reçus tel que : 8 bits de données + 1 bit d'arrêt sans génération/vérification de parité.

• Définit le débit en bauds comme étant le rythme d'horloge aux broches 3 et 4 divisé par 64 (2 MHz/64 = 31,25 kHz, ce qui est le débit spécifié pour MIDI). L'ACIA est maintenant prêt à recevoir et à transmettre des données. Pour le vérifier, lisez le registre d'état avec :

```
PRINT PEEK(56832)
(PRINT?&FEE0)
```

Vous devriez lire la valeur 2 qui indique que le registre de données de transmission (bit 1 mis à 1) et le registre de données de réception (bit 0 mis à 0) sont vides. Puisqu'aucune donnée n'a été reçue et que les interruptions sont invalidées, les autres bits d'état 2 à 7 doivent être à 0.

3. Envoyez un octet du registre de transmission par le câble et dans le registre de réception avec la commande :

```
POKE 56833,X (?&FEE1=X)
```

où X est un nombre compris entre 0 et 255. Cela place une valeur à transmettre dans le registre de données.

4. Pendant le temps nécessaire pour taper la commande suivante, l'octet devrait être reçu. Lisez de nouveau le registre d'état :

```
PRINT PEEK(56832)
(PRINT?&FEE0)
```

La valeur devrait maintenant être 3. Le bit 1 passe immédiatement à 0 après la dernière commande (registre de transmission plein) mais revient à 1 peu de temps après. Le bit 0 est mis à 1, ce qui indique que l'octet a été reçu et peut être lu à partir du registre de données. Notez qu'il y a une possibilité d'erreur : si le bit 0 n'est pas mis à 1, il y a probablement un circuit ouvert dans le chemin de transmission, qui retient l'entrée du récepteur à un niveau élevé l'empêchant d'être détecté.

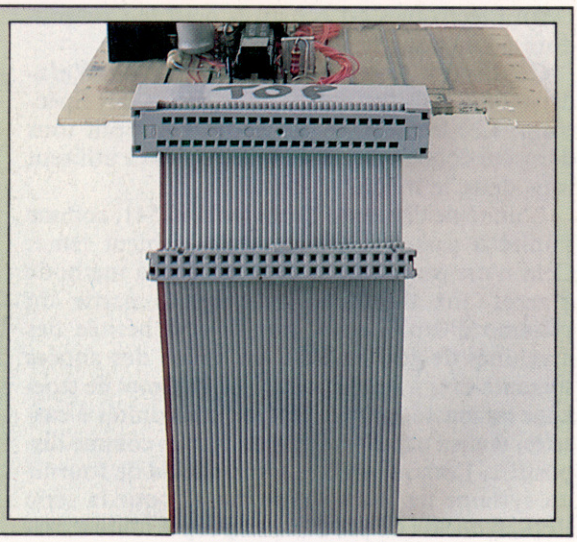
5. Après avoir vérifié la réception d'un octet, nous pouvons vérifier qu'il est bien identique à celui qui a été transmis en lisant le registre de données :

```
PRINT PEEK (56833)
(PRINT ?&FEE1)
```

Cette opération de lecture devrait renvoyer la même valeur X qui a été transmise auparavant. Les étapes 3 à 5 devraient être répétées avec diverses valeurs de X.

Mise au point du ruban

La carte d'interface MIDI peut être reliée au port tube du BBC Micro au moyen d'un câble-ruban à 40 voies, d'un connecteur plat à 40 voies qui se branche dans la carte et d'une prise IDC à 40 voies. Ce câble se branche directement dans le port tube sous le BBC Micro. Le câble et les connecteurs sont faciles à assembler, mais il faut noter l'orientation des deux fiches. Posez le câble à plat et reliez le connecteur plat de façon que la broche 1 (marquée sur le boîtier du connecteur) se situe dans la rangée du bas. Marquez la surface supérieure du boîtier de façon à indiquer l'orientation du connecteur lorsqu'il est utilisé avec la carte d'interface. La prise IDC doit être reliée à l'autre bout du câble, l'encoche rectangulaire vers le haut.



Rapidité de stockage

Avec l'introduction du Quick Data Drive 8500 de Phonemark, qui utilise un nouveau support, les « wafers », le problème du stockage sur le C64 est réellement amélioré.

Comme de nombreux utilisateurs d'ordinateurs domestiques manifestent de plus en plus le désir d'acheter des dispositifs de stockage plus rapides et plus efficaces, les fabricants ont commencé la production d'une vaste gamme de dispositifs destinés aux micros les plus populaires. Parmi ces unités, le Phonemark 8500 Quick Data Drive de Dean Electronics est construit pour le Commodore 64 et pour le Vic-20.

Attaque frontale
Peu de constructeurs ont proposé des unités de stockage en remplacement des unités à cassette et à disquette de Commodore, malgré les critiques dont elles furent l'objet.
Le Quick Data Drive est un système de stockage qui utilise des boucles de bande continue identiques à celles du Rotronics Wafadrive. Le système n'est pas particulièrement bon marché mais, pour la moitié du prix de l'unité de disquette Commodore, il est beaucoup plus rapide. (Cl. Chris Stevens.)

Ce dispositif est un proche parent du Wafadrive de Rotronics conçu pour le Sinclair Spectrum. Les lecteurs de ces deux unités ont tous deux été conçus par BSR Electronics et utilisent tous deux le même support.

L'unité de disquette Commodore 1541, comme l'unité à cassette, est remarquablement lente. Cela n'est pas dû au disque mais à la méthode d'accès aux données. La majeure partie du système d'exploitation du C64 est héritée des machines de gestion PET du milieu des années soixante-dix; à cette époque, les systèmes de stockage de masse, particulièrement les unités à cassette, étaient extrêmement peu fiables comme dispositifs. Lorsque Commodore décida de fournir un système de stockage de masse pour la série PET, il décida de produire sa propre unité à cassette et d'intégrer une série de vérifications pour s'assurer que les données chargées étaient correctes.

Aujourd'hui, la qualité des cassettes s'est grandement améliorée et les longues et complexes vérifications de données sur le Commodore sont devenues inutiles. Plusieurs logiciels commerciaux contiennent leurs propres techniques de chargement qui ont supprimé de nombreuses vérifications et qui accélèrent le chargement, sans perte au niveau de la fiabilité.

Cependant, quand les utilisateurs veulent charger leurs propres programmes, ils ne peuvent profiter de ces techniques et doivent toujours souffrir de la lenteur du système d'exploitation Commodore. Le Quick Data Drive, qui est censé être quinze fois plus rapide que les cassettes normales et plus rapide que l'unité de disquette 1541, est déjà un sérieux concurrent des périphériques Commodore.

Le Quick Data Drive est une unité assez petite. Elle se connecte au Vic-20 ou au Commodore 64 *via* le connecteur plat à cassette. Contrairement au Wafadrive de Rotronics, le Quick Data Drive n'a qu'un seul lecteur. Bien qu'il soit préférable de disposer de deux lecteurs pour pouvoir utiliser simultanément un support programme et un support données, la plupart des utilisateurs se contentent du lecteur unique. Si cela est absolument nécessaire, il est possible de connecter un deuxième lecteur.

Le système d'exploitation

Même si le lecteur se branche dans le port cassette, cela ne veut pas dire que vous ne pouvez pas utiliser une unité à cassette en même temps. Le Data Drive possède un connecteur plat à l'arrière de l'unité, ce qui permet d'y brancher une unité à cassette, ou un second Quick Data Drive connecté en chaîne.

Si le Rotronics Wafadrive et le Quick Data Drive se ressemblent, leur fonctionnement est assez différent. Alors que le Wafadrive possède son propre système d'exploitation en ROM (concurrençant les ROM de l'Interface 1), le système d'exploitation du Quick Data Drive (QOS) est stocké sur le support magnétique lui-même. Pour charger le QOS dans l'ordinateur, vous devez appuyer sur Shift/Run (comme vous feriez pour charger une cassette normale). Quand le message de sollicitation PRESS PLAY TAPE apparaît, vous n'avez qu'à appuyer sur un petit bouton placé à l'arrière du lecteur qui amorce le système. Cela

fait, le QOS l'exécutera automatiquement pour les accès suivants.

Les programmes qui composent le QOS sont chargés dans deux zones de mémoire distinctes. D'abord, les programmes de code machine sont chargés entre les adresses C000 et CFFF en haut de la mémoire (normalement utilisée pour les programmes écrits en langage machine). C'est ce module du QOS qui accélère le chargement.

Bien que le QOS ne mette en œuvre aucune commande qui lui soit propre (utilisant à la place les commandes déjà disponibles dans la ROM du système d'exploitation Commodore), il intercepte les programmes qui gèrent le chargement normal et insère les siennes.

L'autre partie du QOS est l'utilitaire de gestion de fichiers (FMU) qui renferme un certain nombre de programmes bien pratiques. Il est stocké dans la moitié supérieure des 8 K de mémoire situés tout juste sous la ROM BASIC — entre les adresses B000 et AFFF. Les 4 K inférieurs entre A000 et AFFF servent comme tampon de fichier séquentiel utilisé par le FMU.

Puisque le FMU est stocké sous la ROM BASIC, vous ne pouvez utiliser les deux en même temps. Afin de l'appeler de la mémoire, la commande LOAD«FMU» doit être exécutée. Cela crée un chevauchement entre les deux zones de mémoire.

Sans aucun doute, les opérations du Quick Data Drive sont beaucoup plus rapides que les méthodes par cassette. A titre d'exemple, le jeu de simulation New World a été utilisé comme test de performances. Le chargement du programme complet de 25 K nécessitait neuf minutes à partir de l'unité à cassette et une minute et demie à partir de l'unité de disquette; ce chargement s'est effectué en moins de trente secondes avec le Quick Data Drive — une amélioration considérable. Cependant, comme pour les systèmes à boucle sans fin, le temps d'accès dépend grandement de la position de la tête de lecture par rapport au début du programme. Le système d'exploitation du Quick Data Drive trouve les fichiers de données demandées en analysant les blocs initiaux de la bande. Lorsque le support est formaté, le système d'exploitation divise la bande en blocs, chacun ayant une section de nom de fichier qui lui est propre. Quand il doit charger un fichier dans l'ordinateur, le Quick Data Drive recherche d'abord le bloc renfermant la première partie du fichier, le charge et trouve le second. De même, quand l'utilisateur sollicite l'affichage d'un répertoire, le QOS lit chacun des noms de fichiers pendant que la tête de lecture parcourt la bande, et enregistre les blocs qui renferment des fichiers et ceux qui sont libres. Lorsque tous les noms de fichiers ont été lus, le QOS affiche la liste des fichiers ainsi que le nombre d'octets encore libres.

QUICK DATA DRIVE

DIMENSIONS

147 × 118 × 49 mm.

INTERFACES

Prise permettant de le connecter au port cassette du Commodore 64 ou du Vic-20,

FORMAT

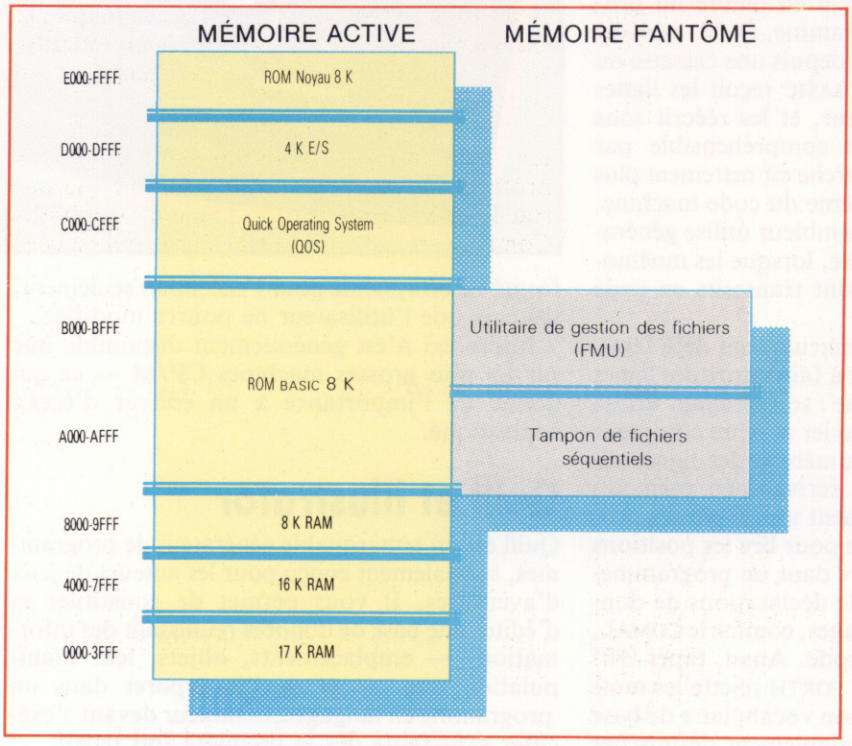
Wafers à boucle de bande continue.

CAPACITÉ

16, 64 et 128 K possibles.

VITESSE

Temps d'accès moyen : fichier de 15 K; 8 fichiers de 128 K; 43 s.



Une mémoire rapide
Le Quick Data Drive ne garde pas son système d'exploitation en ROM, mais le charge en RAM à partir de son support. Les divers composants sont placés dans deux zones de mémoire séparées. Le premier, le QOS, est chargé dans la zone normalement réservée aux programmes écrits en langage machine. L'autre partie du système, l'utilitaire de gestion des fichiers (FMU), est stockée dans la RAM juste derrière la ROM BASIC.
(Cl. Caroline Clayton.)

Utilitaire de gestion de fichiers

L'utilitaire de gestion de fichiers est un ensemble de programmes pilotés par menu; il gère diverses applications, comme le formatage et la lecture du répertoire. Il contient aussi des programmes de copie qui permettent de transférer les données d'une disquette, d'une cassette ou d'un *wafer* vers un wafer de sauvegarde. C'est évidemment une caractéristique importante du système Quick Data Drive, puisque peu de programmeurs envisageraient de passer à ce nouveau moyen de stockage s'ils ne pouvaient transférer les programmes existants sur le nouveau support.

Évidemment, le système comporte ses inconvénients. Bien que les routines de copie fonctionnent très bien avec les programmes BASIC et les fichiers séquentiels, certains programmes (le code machine particulièrement) qui sont chargés dans des zones spécifiques de mémoire posent des problèmes. Cela s'explique par le fait que les deux zones qui sont utilisées par le QOS et le FMU sont aussi des zones de stockage de programmes écrits en langage machine. Par conséquent, lors du chargement des programmes écrits en langage machine, ces derniers écrasent le QOS et bloquent le programme. Cette particularité signifie que, bien que vous puissiez charger vos propres programmes, il n'est pas encore possible de transférer des programmes commerciaux. Si le Quick Data Drive remporte assez de succès commercial, un programme sera certainement créé pour assurer cette fonction. Jusqu'à maintenant, il incombait à Dean Electronics de convaincre les maisons de logiciels de vendre leurs jeux et leurs applications sur wafers. Bien qu'il soit un peu tôt pour se prononcer, les choses s'annoncent bien pour ce produit. Certains jeux, comme Impossible Mission, sont déjà disponibles sur ce nouveau support, mais une base logicielle beaucoup plus importante sera nécessaire.

Code créant du code

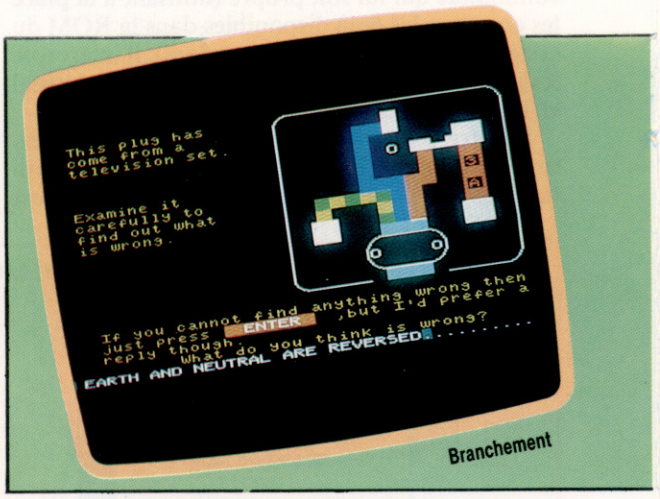
Nous présentons ici un aperçu de quatre logiciels générateurs de programmes et procédons à une étude détaillée du fonctionnement de deux d'entre eux : Sycero et The Last One.

Un programme qui supprime le côté fastidieux de la programmation et qui aboutit à des lignes de code s'exécutant à la perfection, dès la première fois et sans message d'erreur de code ou de syntaxe, peut sembler relever du rêve pour un programmeur.

On peut pourtant dire que toute écriture de programme BASIC met déjà en œuvre un programme qui écrit un programme. Qu'il réside en ROM ou qu'il soit chargé depuis une cassette ou un disque, l'interpréteur BASIC reçoit les lignes tapées par le programmeur, et les réécrit sous forme de code machine compréhensible par l'ordinateur. Et cette démarche est nettement plus rapide que d'écrire soi-même du code machine. La programmation en assembleur utilise généralement un procédé similaire, lorsque les mnémoniques de l'assembleur sont transcrits en code machine.

La plupart des programmeurs sont déjà familiers avec le procédé visant à faire écrire des lignes de code par un programme : soit en ayant utilisé le tampon mémoire du clavier avec un compteur d'incrémentation pour numéroter les lignes du programme, soit en les écrivant en mémoire (POKE). Cela est fréquemment utilisé par les programmeurs code machine pour lire les positions mémoire (PEEK) et les écrire dans un programme-chargeur sous la forme de déclarations de données (DATA). Certains langages, comme le COMAL, dépistent les erreurs de code. Aussi, taper PRNT pour PRINT sera refusé. Le FORTH rejette les mots qui ne figurent pas dans son vocabulaire de base ou qui n'ont pas été préalablement définis par l'utilisateur. Le LOGO est également en mesure de se cantonner à ses propres limitations.

Néanmoins, les caractéristiques que nous venons de voir ne sauraient vous permettre, par exemple, de demander à l'ordinateur d'écrire un programme de calcul de vos impôts. Mais l'importance croissante des programmes qui imitent ou semblent même incorporer un certain degré d'intelligence artificielle rend cette éventualité moins utopique. Ainsi, il existe un programme appelé Microtext, qui est à même de guider un utilisateur non professionnel dans des tâches techniques compliquées. Il peut également générer des outils pédagogiques interactifs, des programmes de dépistage d'erreurs, et du logiciel fournissant à la demande des informations ou des questionnaires. En utilisant un système auteur pour générer une application, avec un éditeur d'écran actif disponible au cours des tests, le logiciel obtenu peut être « publié » et transformé en un système pour l'exécution seulement, système que l'utilisateur ne pourra modifier.



Branchement

Microtext n'est généralement disponible que sur les plus grosses machines CP/M — ce qui donne de l'importance à un éditeur d'écran sophistiqué.

Quill et Illustrator

Quill est un remarquable générateur de programmes, spécialement conçu pour les auteurs de jeux d'aventures. Il vous permet de constituer et d'éditer une base de données réunissant des informations — emplacements, objets, leur manipulation, etc. — et de l'incorporer dans un programme en langage assembleur devant s'exécuter sans faute dès la première fois (RUN).

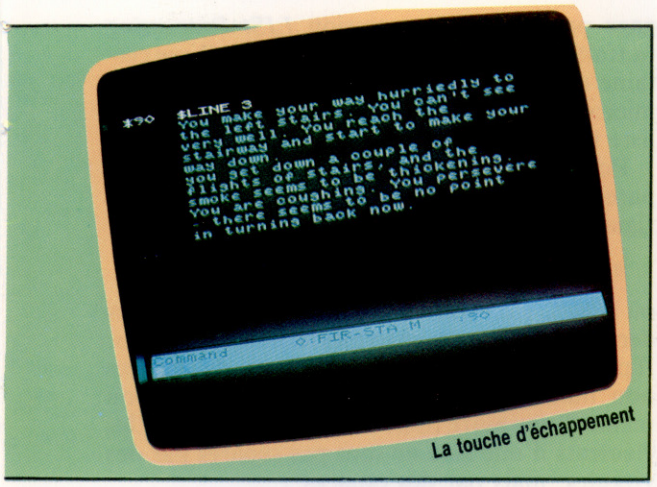
Au Quill est venu dernièrement s'ajouter Illustrator, qui allie du graphisme aux textes. Malgré son faible coût, Quill est suffisamment efficace pour avoir permis l'écriture de jeux d'aventures disponibles dans le commerce.

Même si les deux logiciels suivants sont bien techniquement des générateurs de programmes, ils ne sont pour l'industrie de la Micro que deux programmes qui écrivent effectivement d'autres programmes : à savoir, le logiciel un peu présomptueux The Last One (« le dernier », sous-entendu « le dernier programme dont vous aurez besoin »...), et le plus puissant (même s'il est moins convivial) Sycero.

TLO, comme l'appellent maintenant ses distributeurs, fut conçu pour des machines CP/M de 64 K. C'est évident si l'on considère les manifestations de débordement de la capacité des disques qu'il suscite (ce qui fait également de WordStar une tortue). Il est maintenant disponible pour

les machines MS/DOS d'un minimum de 256 K de RAM. Mais, même s'il permet des configurations supplémentaires sophistiquées, cela n'accélère pas pour autant l'exécution du programme.

Les auteurs de TLO affirment avoir incorporé un certain degré d'intelligence artificielle dans le programme. De fait, si on le charge avec un autre BASIC que celui auquel il s'attend, il identifie le système d'exploitation à disque par un message d'erreur approprié.



Jeux de génération
Ces photos d'écran montrent le genre de programmes que peut produire le générateur Microtext. A gauche, le programme du « test de la prise électrique » pose à l'utilisateur des questions. A droite, le programme teste vos connaissances sur la procédure à suivre en cas d'incendie dans un hôtel. Vous remarquez que ces deux programmes supposent que l'ordinateur soit interactif avec l'utilisateur, afin d'évaluer les réponses de ce dernier selon la base de connaissances de sa mémoire. Ce type d'application convient tout à fait à un générateur de programmes.

Bien que similaires en apparence (ils sont tous les deux des générateurs de programmes), Sycero et TLO sont en fait très différents au niveau de la conception et de la portée. En outre, leur approche du système de la simplification des besoins de l'utilisateur n'est pas identique. En effet, tout générateur de programmes doit être à même d'exprimer de manière claire les demandes de l'utilisateur, afin de produire un code utilisable et transférable. Ils sont également semblables en ce qu'ils génèrent tous les deux des programmes en BASIC, et qu'ils sont particulièrement bien adaptés à la saisie, au traitement, au stockage, à la restitution et à l'impression des données.

Ils diffèrent par ailleurs des programmes qui effectuent des opérations sur des bases de données, car le code résultant est exécutable indépendamment du système générateur.

D'une certaine manière, TLO est proche des besoins de l'utilisateur d'aujourd'hui qui préfère « penser à l'écran » plutôt que d'avoir à faire les calculs et études préliminaires sur le papier.

Mais les logiciels Sycero et TLO méritent beaucoup plus d'attention que cela : le plan que suggère Sycero insiste sur une préparation minutieuse en sept points :

— plan ;
— spécification du système ;
— dessin des écrans ;
— vérification des données ;
— définition du programme ;
— génération d'états ;
— génération du programme.

En outre, pour que cela soit bien clair, Sycero et TLO disent explicitement dans leurs manuels : l'organisation et la préparation préliminaires d'un

The Last One : pour Apple II et Apple IIe, et pour machines CP/M-80 et MS-DOS.

Microtext : pour le Tatung Einstein et les machines CP/M-80.

Quill : pour le Spectrum, le Commodore 64, l'Amstrad 464 et l'Oric.

Sycero : pour machines MS-DOS.

plan sont essentielles à une bonne programmation. Voici d'abord ce que dit le manuel de Sycero : « La correction d'erreurs dues à une mauvaise conception du programme peut se révéler plus longue que l'écriture soigneuse initiale du système. Il est toujours tentant de bâcler en cinq minutes l'étude du problème, pour se précipiter sur Sycero et construire le système par à-coups : ce n'est pas la bonne approche. [...] Vous devez toujours commencer par recenser de manière exhaustive toutes les opérations que vous voulez faire effectuer par l'ordinateur dans le sens de ce que vous voulez voir à l'écran (information permanente) et de ce que vous désirez obtenir comme états (recopies d'écran et rapports). Et ce n'est qu'une fois défini ce que vous voulez obtenir du système que vous pouvez réfléchir aux informations à réunir en vue de ce résultat. »

TLO est moins explicite mais parle bien de la même chose : « Commencez par planifier le déroulement d'ensemble de votre programme, et prévoyez les erreurs possibles de l'utilisateur final. Assurez-vous alors de leur dépistage infaillible. Vous devez en fait concevoir votre programme aussi à fond que toute autre tâche. En règle générale, il est plus facile de commencer par écrire un certain nombre de courts programmes, liés entre eux par un simple menu qui les appelle selon les besoins du traitement.

Vous découvrirez ensuite que cette approche modulaire permet de planifier et de créer facilement des programmes importants et complexes. »

Ces deux logiciels ne génèrent pas des programmes tout faits. Ils ne créent pas de jeux, pas même un jeu d'aventures avec seulement du texte, et encore moins un tableur ou un logiciel de traitement de texte.

Leur force tient à ce qu'ils vous permettent de manipuler pratiquement toutes sortes de données, d'effectuer des calculs sur ces données (calculs de tous types, depuis l'extraction du taux de T.V.A. d'une somme brute, jusqu'à la résolution d'une équation d'ingénierie longue et complexe), de stocker des données, les restituer et les imprimer. A cet égard, ces deux générateurs sont extrêmement puissants.

On peut discuter pour savoir s'ils sont aussi efficaces qu'un programme de gestion de base de données tel que dBase II par exemple ; il est indéniable qu'ils sont à la fois plus faciles d'emploi et davantage conviviaux que ces programmes hautement performants, mais rébarbatifs et difficiles à l'utilisation. Nous verrons l'efficacité des générateurs de programmes et leur facilité d'utilisation.

Entrez les listes

Fondé sur des listes pouvant représenter des données ou des fonctions, le lisp, très répandu dans l'intelligence artificielle, permet d'adapter le langage à pratiquement toutes les applications.

Le LISP a acquis une grande notoriété ces dernières années du fait de sa participation aux programmes de recherche et de développement en matière d'intelligence artificielle. Comme l'intérêt pour ce langage s'accroissait, on s'aperçut qu'il était également un langage universel aux applications multiples.

Cette prise de conscience suscita un nombre considérable de versions LISP dans des applications allant des systèmes d'exploitation et des compilateurs aux jeux d'aventures. Cette prolifération a pour conséquence malheureuse le manque d'unité d'une version à l'autre du LISP (c'est le cas pour d'autres langages). Malgré cela, la plupart des dialectes LISP mis en œuvre sont relativement standard, grâce à la structure simple du langage. Le transfert de LISP d'une machine à une autre est assez aisé.

Un des problèmes que vous rencontrerez, en revanche, lors de l'utilisation du LISP, est l'absence de fonctions ou d'instructions. Cela est dû au manque de standard officiel; le programmeur ne doit garder que les fonctions qu'il estime essentielles. Cependant, ainsi que nous allons le voir, il est très facile d'enrichir ce langage, en ajoutant les instructions dont on peut avoir besoin.

Versions LISP
Malgré l'intérêt croissant que connaît LISP, il existe peu de versions bon marché pour les utilisateurs de micros. Le LISP d'Acornsoft est disponible pour le BBC Micro et l'Acorn Electron. Les interpréteurs fondés sur CP/M; devraient pouvoir être exploités sur les micros hôtes acceptant CP/M : mais les utilisateurs d'Amstrad ne pourront pas les utiliser étant donné le manque de place mémoire (le minimum nécessaire étant de 48 K). Toolworks est probablement la version la plus intéressante, avec un espace mémoire d'environ 3 600 positions de listes, et 11 000 noms atomes dans 48 K.

Une structure syntaxique unique

Le LISP est entièrement différent des langages plus courants tels que le PASCAL, le FORTRAN ou le BASIC. La structure de sa syntaxe est tout à fait unique, simple et cohérente. Elle fait de ce langage l'outil idéal pour le traitement des données et la résolution des problèmes.

Sa structure de listes inhérentes s'adapte facilement aux principales structures d'information de la programmation, et autorise l'utilisation du langage pour les opérations de recherche et de tri, pour les fonctions arithmétiques et même pour les jeux. En outre, la plupart des micro-ordinateurs disposent d'instructions spécifiques destinées à exploiter leurs propres caractéristiques graphiques et sonores.

C'est certainement le cas avec le LISP d'Acornsoft, que nous utiliserons dans cette suite d'articles. Ne vous inquiétez pas si vous disposez d'une autre version du langage, vous découvrirez que tout est directement transférable.

Le fondement de LIST, vous vous en doutez, est la structure de données list. Le nom du langage provient de l'expression *LISt Processing* (« traitement de listes »). Une liste est d'une certaine manière assimilable à un tableau pour les micros. Cependant, contrairement à celui-ci, une liste n'a pas de longueur spécifique.

Pour écrire une liste, il suffit de mettre entre parenthèses les éléments que l'on désire y voir figurer :

```
(a b c d e...)
```

où a, b, c, etc. sont les éléments de la liste. Ces derniers sont appelés en termes techniques *atomes*. Il peut s'agir de données numériques, de données non numériques (caractères ou variables), ou même d'une autre liste. Vous remarquez que les éléments d'une liste ne sont pas séparés par des virgules, mais par des espaces.

Les opérations sur les listes se font par des fonctions. On peut comparer leur démarche à celle d'une fonction BASIC (DEF FN commande) qui effectue une opération sur ses arguments pour obtenir un résultat. Une fonction s'écrit sous la forme suivante :

```
(fonc a b c d...)
```

où fonc est le nom de la fonction, et où a, b, c, d sont ses arguments. Comme vous pouvez le constater, la fonction semble être le premier élément de la liste.

Ainsi, on écrirait de la manière suivante la liste contenant les six premiers nombres premiers :

```
(1 2 3 5 7 11)
```

En BASIC, si nous voulions les additionner entre eux, nous écririons :

```
1 + 2 + 3 + 5 + 7 + 11
```

Avec LISP, nous utilisons la fonction PLUS :

```
(PLUS 1 2 3 5 7 11)
```

ce qui donne le résultat 29. Une des caractéristiques de la fonction PLUS, est de pouvoir prendre un nombre indéfini d'arguments. Aussi, nous pourrions très bien écrire :

```
(PLUS 1 2 3 (PLUS 5 7 11))
```

La fonction la plus interne est évaluée en premier, et donne 23. Ce résultat intermédiaire est alors utilisé comme quatrième argument de la fonction la plus externe, pour obtenir le résultat final : 29. Il faut noter ici la facilité avec laquelle les fonctions s'imbriquent. En imbriquant une autre fonction, SETQ, nous pouvons affecter le résultat à la variable A :

```
(SETQ A (PLUS 1 2 3 5 7 11))
```

où SETQ a deux arguments — la variable A et une fonction dont l'évaluation donne 29. Bien sûr, SETQ est elle-même une fonction et doit donner un résultat (ici la valeur 29). Aussi, l'affectation suivante, illégale avec la plupart des BASIC,

```
LET B = 1 + 2 + 3 + 5 + 7 + 11
```

pourrait s'écrire ainsi avec LISP :

```
(SETQ B (SETQ A (PLUS 1 2 3 5 7 11)))
```

En introduisant la fonction TIMES, nous pouvons assigner, par exemple, deux fois la valeur de A à la variable B :

```
(SETQ (TIMES 2 (SETQ A (PLUS 1 2 3 5 7 11))))
```

Cette expression évalue d'abord PLUS à 29, valeur affectée à la variable A par la fonction SETQ la plus interne. Cette dernière donne à son tour 29, qui devient le deuxième argument de la fonction TIMES. Le nouveau résultat, 58, est alors transmis à la fonction SETQ la plus externe, afin de placer cette valeur dans la variable B. L'ensemble de l'expression donne le résultat 58, utilisable par d'autres fonctions, et ainsi de suite.

L'importance des parenthèses

On remarque que les parenthèses sont vraiment très nombreuses dans le dernier exemple. C'est une caractéristique de LISP, et il peut être fastidieux de les recenser toutes. Pourtant, si le programme est bien conçu, ces parenthèses se « surveillent » entre elles. En outre, certains systèmes LISP vous aident en tenant la comptabilité des parenthèses restantes. Ainsi, avec Acornsoft LISP sur BBC Micro, le nombre de flèches en début du message-guide de ligne indique le nombre de parenthèses que vous devez encore ajouter pour terminer l'expression.

Nous avons introduit dans la dernière expression la fonction TIMES qui comportait deux arguments : l'entier 2 et une expression liste qui avait été préalablement évaluée à 29. Néanmoins, la fonction TIMES, tout comme PLUS, peut avoir un nombre variable d'arguments. Les expressions suivantes sont toutes légales :

```
(TIMES 1 2 4 8 16)
(TIMES 1 2 4 8 (TIMES 4 4))
(TIMES 1 2 (TIMES 2 2) (TIMES 2 4) (PLUS 8 8))
```

et donneraient toutes le résultat 1024.

En réalité, la plupart des versions LISP sont limitées quant au nombre d'arguments que ces types de fonctions peuvent accepter. Par exemple, Acornsoft LISP a une limite de vingt-huit arguments, et d'autres versions sont encore plus limitées.

Nous avons donc vu qu'une liste LISP est constituée simplement d'un ensemble d'éléments, dont le premier est la fonction à exécuter, les éléments suivants étant les arguments de la fonction. Ces arguments peuvent à leur tour être des listes dont le premier argument sera la fonction donnant un résultat. La question de savoir quoi faire se pose si nous ne voulons pas de fonction dans une liste. Nous pouvons parfaitement dresser une liste d'éléments de données, qui sera utilisée par exemple en tant que titre. Nous ne pouvons écrire :

```
(COURS D'INFORMATIQUE AVANCE)
```

sous la forme d'une liste de quatre éléments de données. LISP essaierait en effet d'évaluer COURS comme un nom de fonction, avec deux arguments qu'il prendrait pour des variables. Nous pouvons demander à LISP de ne pas évaluer une expression en faisant précéder cette dernière d'un guillemet simple ('). Aussi, l'expression ci-dessus s'écrit :

```
'(COURS D'INFORMATIQUE AVANCE)
```

Nous pouvons alors affecter cette liste à une variable :

```
(SETQ MAG '(COURS D'INFORMATIQUE AVANCE))
```

qui assignera une liste de trois éléments non numériques à la variable MAG.

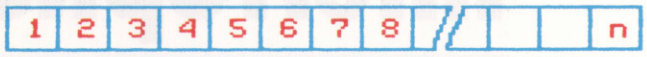
Remarquez qu'il n'y a pas de type variable avec LISP. Si nous utilisions le BASIC, une variable entier s'écrirait A% ; une variable réel, A ; et une variable chaîne, A$. Avec le LISP, les variables ne sont pas différenciées de la sorte, aussi les expressions suivantes sont-elles toutes légales :

```
(SETQ A 3)                          A = l'entier 3.
(SETQ A 'COURS)                     A = la chaîne 'COURS'.
(SETQ A (PLUS 2 4 8))               A = résultat de 2 + 4 + 8.
(SETQ A '(1 2 4 8))                 A = la liste (1 2 4 8).
(SETQ A '(COURS D'INFORMATIQUE))    A = la liste (COURS D'INFORMATIQUE).
(SETQ A B)                          A = valeur de la variable B.
```

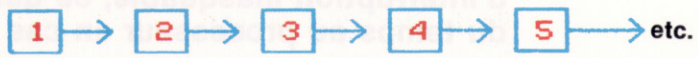
Bien que les types entier et chaîne soient relativement standards, peu de micros supportent une version de LISP effectuant les calculs en virgule flottante. Dans la plupart des cas, l'arithmétique des entiers naturels est suffisante ; mais, au cas improbable où le calcul en virgule flottante serait nécessaire, il pourrait sans nul doute être simulé au moyen des entiers standards.

Il peut sembler difficile à ce stade de notre exposé d'apprécier l'utilité du LISP. Nous verrons dans le prochain article les fonctions LISP qui manipulent leurs arguments de données. Nous verrons également comment les principes de récurrence (dont nous avons déjà parlé) sont mis en œuvre pour faire de LISP un langage extrêmement puissant.

TABLEAU (DIMensionné à n)



LISTE (pas de longueur déterminée)



Listes contre tableaux
Les listes ont deux avantages principaux sur les tableaux. D'abord, la mémoire n'a pas besoin d'être réservée préalablement ; ensuite, une liste est une structure dynamique, ce qui revient à dire qu'elle n'a pas besoin d'une longueur déterminée et peut s'étendre ou diminuer pour suivre les besoins des données pendant l'exécution du programme. En outre, les listes se prêtent plus facilement aux procédures récursives rencontrées fréquemment en programmation d'intelligence artificielle.

Interruptions

L'UC Z80 offre au programmeur en langage machine trois modes d'interruption masquable, ce qui vous permet d'« emprunter » du temps au processeur en cas de besoin.

Il faut prendre beaucoup de précautions en utilisant les possibilités offertes par les interruptions du système d'exploitation du Spectrum. Si par mégarde on les invalide, le fonctionnement de la machine peut être sérieusement affecté — le clavier peut ne pas être lu, par exemple, ou le système peut « se planter » pendant l'exécution de programmes BASIC.

L'unité centrale Z80, au cœur du Spectrum, répond à deux types différents d'interruptions : masquables et non masquables (NMI). La différence entre elles est toute simple. Nous pouvons programmer l'UC pour ignorer un signal d'interruption masquable, mais le processeur répondra toujours à une interruption non masquable.

Les interruptions NMI sont un peu difficiles à utiliser sur le Spectrum, parce qu'il semble que la routine ROM manipulant les NMI contienne des bogues. L'intention des concepteurs du SE était de permettre à l'utilisateur de spécifier une adresse dans les variables système inusitées aux emplacements &5CB0 et &5CB1, et celle-ci devait être sautée chaque fois que l'UC recevait une impulsion NMI. Toutefois, la réponse habituelle du Z80 consiste à exécuter le langage machine commençant à l'adresse &66, ce qui provoque généralement une complète réinitialisation du système. Par conséquent, nous concentrerons notre attention sur l'utilisation des interruptions masquables.

Il y a plusieurs façons pour l'UC Z80 de répondre à une interruption masquable, et ces alternatives sont appelées *modes* d'exploitation des interruptions. Ici, nous ne traiterons que les modes significatifs pour le SE Spectrum.

A l'initialisation — qui a lieu quand vous mettez la machine sous tension ou émettez une commande NEW — l'UC met le mode interruption 1, ou IM1. Le Spectrum tourne normalement dans ce mode d'interruption. Les impulsions d'interruption sont fournies à l'UC par l'ULA (*Uncommitted Logic Array,* « tableau de logique non commutée ») à la vitesse de 50 interruptions à la seconde. En mode IM1, l'UC exécute une instruction RTS &0038 en recevant un signal d'interruption. Il en résulte un saut aux routines qui lisent le clavier et mettent à jour le compteur FRAMES, situé aux emplacements RAM 23672 à 23674. (Ces trois octets forment un compteur à 24 bits qui est donc mis à l'heure toutes les 20 μs.)

Après avoir interprété une ligne de BASIC, l'interpréteur BASIC attend une interruption avant de passer à l'interprétation de l'instruction suivante. Cela implique que, si vous invalidez les interruptions, l'exécution du programme BASIC s'arrêtera.

Les interruptions non masquables seront ignorées par l'UC après exécution d'une instruction DI (*Disable Interrupt,* « invalide interruption »), et elles sont revalidées par une commande EI (*Enable Interrupt,* « valide interruption »).

Différentes routines du SE Spectrum demandent que des interruptions soient invalidées lorsqu'elles sont exécutées. Ce sont généralement des routines dépendant du temps, telles que la routine BEEP du générateur de son et les routines de sauvegarde et de chargement. Des interruptions peuvent aussi être temporairement invalidées par l'imprimante ZX, l'Interface 1 ou le Microdrive. Évidemment, le SE revalide les interruptions dès qu'il a terminé la routine.

Un résultat pratique de différentes routines invalidant des interruptions est que, pendant l'exécution de ces routines, le compteur FRAMES n'est pas incrémenté. Cela causera une « perte de temps » durant l'exécution de ces opérations. Comme une interruption ne peut se produire quand l'UC exécute un de vos programmes, il est recommandé d'invalider les interruptions pour

Anticiper toutes les possibilités

Si l'on suppose que le bus de données du Spectrum contient 255 lorsqu'une interruption est générée, certains périphériques (tels que l'interface manche à balai Kempston) peuvent cependant modifier cette valeur. Le moyen le plus simple de contourner ce problème consiste à s'assurer que vous n'utilisez pas d'interruption en mode 2 (IM2) en rapport avec des périphériques.

Une autre méthode, qui permettra toutes les valeurs possibles sur le bus de données, consiste à remplir une page de mémoire avec des valeurs identiques. Le numéro de page est alors chargé dans le registre I avant de sélectionner IM2.

Lors d'une interruption, le Z80 cherche alors une adresse d'un programme de service interruption à partir d'une position dans une page spécifiée — la position exacte devant être déterminée par la valeur sur le bus de données.

D'abord, nous chargeons le registre I avec la valeur &FC, et remplissons les octets &FC00 à &FD00 avec la valeur &FB. Puis nous sélectionnons IM2. Si le bus de données contient &C3 lorsqu'a lieu l'interruption suivante, le Z80 prend l'adresse de routine service interruption de l'emplacement &FCC3 — la partie &FC de l'adresse ayant été fournie par le registre I.

L'adresse recueillie sera, bien sûr, &FBFB — puisque tous les 256 octets de la page FC (plus l'octet zéro de la page &FD) ont été mis à la même valeur.

Il y a deux points significatifs à noter ici. Le premier est que votre programme de service interruption (ISR) sera toujours à une adresse dont les octets lo et hi sont identiques (par exemple : &C4C4 ou &FDFD). Le second point est qu'il ne faut pas oublier de tenir compte du fait que le bus de données peut en fait contenir &FF quand une interruption a lieu. Dans ce cas, le Z80 regarde la routine ISR aux adresses nnFF (octet lo) et (nn+1)00 (octet hi), où nn est la valeur du registre I. C'est pourquoi vous devez penser à fixer le premier octet de la page suivante.

toute partie de votre programme dans laquelle une synchronisation précise est importante. Toutefois, il est *vital* de les revalider avant de faire le retour au BASIC !

En pratique, il est nécessaire de changer le mode d'interruption à 1, ce qui permet une plus grande versatilité. Le mode d'interruption le plus utile est IM2, qui est plus compliqué que le mode d'interruption 1. Alors que IM1 a toujours pour effet un saut à l'adresse &0038 (en utilisant l'instruction RST &0038), IM2 peut servir à sauter à une routine en un autre endroit de la mémoire. L'adresse du saut de l'UC est spécifiée en utilisant ce qu'on appelle un *vecteur d'interruption*.

Dans une interruption vectorisée, le vecteur contient l'adresse de la *routine de service d'interruption* qui doit être exécutée lorsqu'une interruption masquable a lieu. L'UC sait où est situé le vecteur en mémoire, en se servant d'un registe spécial dans le Z80, appelé *registre I*. L'adresse de la routine vecteur d'interruption est trouvée en combinant le contenu du registre I avec celui du bus de données à l'instant où a lieu l'interruption. Dans certains systèmes, le dispositif causant une interruption mettra un octet sur le bus de données pour dire à l'UC ce qui a causé l'interruption.

L'ULA Sinclair ne met cependant pas de valeur sur le bus de données lorsqu'il envoie un signal d'interruption au Z80, mais la manière dont le matériel Spectrum est aménagé fait en sorte que le bus de données contienne la valeur 255 lorsqu'aucune autre entrée n'y est appliquée. Le registre I fournit donc l'octet hi de l'adresse 16 bits du vecteur d'interruption, et la valeur sur le bus de données (dans ce cas &FF) fournit l'octet lo de l'adresse. Ainsi, le vecteur d'interruption sera toujours placé à une limite de page en mémoire, l'octet lo de la routine service d'interruption étant contenue à l'adresse &nnFF et l'octet hi à l'adresse &(nn+1)00 (où nn est le contenu du registre I).

Par exemple, si le registre contient la valeur &FB, l'adresse du vecteur sera en &FBFF. L'octet lo du vecteur, en &FBFF, contiendra l'octet lo de l'adresse de la routine service d'interruption, et l'octet hi du vecteur, à l'adresse &FC00, contiendra l'octet hi de l'adresse ISR.

Il y a des restrictions sur l'endroit où l'ISR peut être situé en mémoire — les 16 premiers K, par exemple, sont assignés à la ROM, c'est pourquoi ils ne peuvent être utilisés. Ensuite, des problèmes matériels empêchent l'exploitation correcte du SE avec une valeur de registre I comprise entre 64 et 127.

Considérons maintenant comment le Z80 est mis en mode d'interruption 2, et fixons le vecteur d'interruption à l'adresse de l'ISR à utiliser :

```
F3          di
210000      ld   hl,ADDRESS          ;get ISR address in HL
22FFFB      ld   (#FBFF),hl          ;get address into vector
3EFB        ld   a,#FB               ;hi-byte of vector...
ED47        ld   i,a                 ;...into I register
ED5E        IM   2                   ;set int mode 2
FB          ei                       ;re-enable interrupts
C9          ret                      ;back to BASIC
```

Naturellement, ce listage suppose qu'il y a une routine en ADDRESS pour manipuler les interruptions — sinon le programme risque de « se planter ». Le programme en assembleur ci-dessous modifie le mode d'interruption en IM2 et a pour effet que l'UC exécute la routine à ADDRESS à chaque interruption (ce qui, dans ce cas, accomplit les fonctions usuelles du Spectrum en mode d'interruption normal).

```
                org  60000               ;specify start add
      vector: equ  #FEFF                 ;FEFF is vector add
216FEA change:  ld   hl,addres           ;get address into HL
22FFFE          ld   (vector),hl         ;set up vector
F3              di                       ;disable interrupts
3EFE            ld   a,#FE               ;set up the...
ED47            ld   i,a                 ;...I register
ED5E            im   2                   ;change int mode
FB              ei                       ;re-enable interrupts
C9              ret                      ;back to BASIC
F3      addres: di                       ;turn off interrupts
FF              rst  #38                 ;normal IM1 procedure
FB              ei                       ;re-enable interrupts
C9              ret
```

L'appel de routine à l'adresse CHANGE fixera le nouveau mode d'interruption et le vecteur. Cela fait, le programme à ADDRESS sera exécuté tous les cinquantièmes de seconde. Pour le moment, ce programme ne fait rien de particulièrement utile, mais nous donnerons brièvement quelques fonctions pratiques.

S'il est nécessaire de changer le mode d'interruption pour revenir au mode normal au cours du programme, une routine telle que celle-ci peut être utilisée à cette fin :

```
F3          di
3E3F        ld   a,#3f               ;reset I register...
ED47        ld   i,a                 ;...to its usual value
ED56        im   1                   ;normal mode
FB          ei                       ;re-enable interrupts
C9          ret
```

Pour que le programme de service d'interruption puisse exécuter votre propre code, nous traitons simplement le code comme un sous-programme et l'appelons (CALL) :

```
F3      addres: di
F5              push af              ;save registers...
C5              push bc
D5              push de
E5              push hl
FF              rst  #38             ;call normal ISR
F3              di                   ;ISR did EI, so DI again
CD0000          call PROG_ADD        ;call our own routine
E1              pop  hl              ;restore registers...
D1              pop  de
C1              pop  bc
F1              pop  af
FB              ei
C9              ret
```

Évidemment, si votre programme de service d'interruption est long, il ralentira légèrement le Spectrum, ce qui affectera l'exécution du programme et la vitesse à laquelle FRAMES sera incrémenté. Enfin, n'oubliez jamais d'invalider des interruptions pendant votre routine, afin d'éviter une seconde interruption au milieu de la première.

Programme musique d'interruption

Le programme suivant montre comment on peut utiliser des interruptions pour produire une « musique de fond » sur Spectrum. Le Compilateur données notes est un programme BASIC qui vous permet d'écrire vos propres airs, tandis que le listage d'assemblage (ou le chargeur BASIC si vous n'avez pas d'assembleur) vous permettra de fixer la routine de musique commandée par interruption et de lui faire passer les données compilées par le Compilateur données notes.

En exécutant (RUN) le Compilateur données notes, on vous présente un menu de trois options. C-COMPILE, s'il est sélectionné, modifiera l'air (contenu dans les instructions DATA, lignes 10 à 900) en un bloc de code qui peut être utilisé par la routine d'interruption. P jouera l'air afin que vous puissiez décider quel changement vous désirez apporter s'il y a lieu. R vous ramènera au BASIC de sorte que vous puissiez sauvegarder (SAVE) les données compilées ou changer les données musique dans les instructions DATA.

Notez que si vous utilisez l'option C, on vous demandera de préciser où vous voulez stocker les données notes en mémoire. Assurez-vous que vous les stockez au-dessus de RAMTOP, valeur que vous aurez modifiée préalablement pour faire de la place pour les données. Une fois les notes compilées et stockées, l'adresse de base et la longueur du code seront affichées. Il faut que vous en preniez note parce que cette information vous servira par la suite. Pour sauvegarder le code, retournez au BASIC et entrez : SAVE «NOTEDATA» CODE (adresse de base), [nombre d'octets de longueur]. Enfin, si vous voulez utiliser vos propres mélodies, il suffit de supprimer les lignes 10 à 90 et de mettre vos propres données n'importe où entre les lignes 10 et 900.

Tapez le listage d'assemblage à l'aide d'un assembleur et sauvegardez-le sur bande. Si vous n'avez pas d'assembleur, utilisez le chargeur BASIC.

La routine a une adresse de base de 65021 ; aussi, avant de charger le code en mémoire, assurez-vous que vous avez abaissé RAMTOP en conséquence (CLEAR 65000 fera l'affaire). Après avoir assemblé et chargé le code, vous aurez à charger le code compilé par le Compilateur données notes (ci-contre). N'oubliez pas d'abaisser encore RAMTOP pour faire de la place aux données notes. Le programme musique d'interruption doit savoir la position exacte des données notes à jouer avant de pouvoir fonctionner. Vous pouvez faire passer cette information à la routine à l'aide du petit programme BASIC suivant :

```
10 LET L=65152: LET V=(*adresse de base de
données notes*)
20 POKE L+1, INT(V/256): POKE L,
V-PEEK(L+1)*256
```

Cela fait, RAND USR 65041 fixera IM2 et commencera à jouer votre mélodie. RAND USR 65071 sélectionnera IM1 et arrêtera la musique.

Compilateur données notes

```
  1 REM >>> INTERRUPT MUSIC <<<
  2 REM >>Note Data  Compiler<<
  3 REM
  6 REM *NOTE DATA IN STANDARD SPECTRUM BEEP FORM
AT *
  7 REM
  9 RESTORE
 10 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,
1,9,2,10,1,10,2,10,1,12,5,14
 20 DATA 1,14,2,7,1,7,1,7,1,6,1,7,2,16,1,14,2,14,
1,10,2,9,1,9,2,9,1,10,5,12
 30 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,
1,12,2,11,1,19,2,19,1,19,5,19
 40 DATA 1,17,2,16,1,19,1,19,1,18,1,19,2,14,1,19,
1,19,1,18,1,19,2,12,1,11,2,12,1,11,3,12
 50 DATA 3,10,1,9,1,8,1,9,2,14,1,12,3.2,9,3.2,5,3
.2,2,3.2,7,5,5
 60 DATA 1,5,1,7,1,9,1,10,2,16,1,14,3.2,12,3.2,17
,3.2,16,3.2,14,5,12
 70 DATA 1,12,2,14,1,14,1,14,1,13,1,14,3,16,3,9
 80 DATA 2,17,1,17,1,19,1,17,1,19,5,21
 90 DATA 1,21,2,19,1,17,2,14,1,10,3.2,9,3.2,5,3.2
,7,3.2,4,5,5
997 REM
998 REM *DO NOT ERASE LINE 999*
999 DATA 255,255
1000 REM
2000 REM **** MENU ****
2010 BORDER 1: PAPER 1: INK 6
2020 CLS
2030 PRINT AT 5,4;"INTERRUPT MUSIC COMPILER";AT 10
,5;"C -- Compile note data";AT 12,5;"P -- Play tun
e in beeps";AT 14,5;"R -- Return to BASIC"
2040 LET a$=INKEY$
2050 IF a$="c" OR a$="C" THEN  GO TO 5000
2060 IF a$="p" OR a$="P" THEN  GO TO 3000
2070 IF a$="r" OR a$="R" THEN  CLS : STOP
2080 GO TO 2040
3000 RESTORE
3010 READ d,f: IF d=255 AND f=255 THEN  GO TO 2000
3020 BEEP d/10,f-6: GO TO 3010
5000 RESTORE : CLS : INPUT "BASE ADDRESS OF NOTE D
ATA? ";dd1: POKE 23301,INT (dd1/256): POKE 23300,d
d1-(256*PEEK 23301): CLEAR dd1-1: LET dd1=PEEK 233
00+256*PEEK 23301
5005 LET f=0: LET d=dd1
5010 READ delay,pitch
5015 LET delay=INT (delay*6)
5020 LET freq=(1.0594631^pitch)*256
5030 LET bip=INT ((437500/freq)-30.125)
5035 IF delay=255*6 THEN  POKE d,255: GO TO 6000
5037 LET f=f+1: PRINT AT 10,10;"NOTE )";f
5040 POKE d,delay
5060 POKE d+1,bip-(INT (bip/256)*256)
5070 POKE d+2,INT (bip/256)
5080 LET d=d+3
5090 GO TO 5010
6000 PAUSE 50: CLS : LET len=(d+2)-dd1
6010 PRINT "LOCATION OF NOTE DATA IN MEMORY=";dd1'
''"NUMBER OF BYTES OF NOTE DATA....";len
6020 INPUT "PRESS 'ENTER' TO RETURN TO MENU "; LIN
E a$: GO TO 2000
```

Chargeur basic

```
 10 RESTORE 9000
 20 CLEAR 64000
 30 LET cqs=0: FOR f=65021 TO 65156
 40 READ dat: POKE f,dat
 50 LET cqs=cqs+dat
 60 NEXT f
 70 IF cqs<>18850 THEN  PRINT "ERROR IN DATA !!!"
: STOP
 80 PRINT "OK NO PROBLEMS IN DATA"
 90 STOP
9000 DATA 0,0,221,229,229,197,213,245,195,52,254,2
41,209,193,225,221,225,195,56,0,33,255,253,34,253,
253,243,62,253,237,71,237,94,42,128,254,34,126,254
,62,1,50,130,254,61,50,131,254,251,201,243,237,86,
251,201,58,130,254,254,0,202,8,254,58,131,254,254,
0,202,75,254,61,50,131,254,195,8,254,42,126,254,12
6,254,255,202,108,254,50,131,254,35,126,35,95,126,
35,34,126,254,103,107,17,4,0,205,181,3,243,195,8,2
54,42,128,254,34,126,254,62,1,50,130,254,61,50,131
,254,195,8,254,0,0,0,0,0,0,0,
```

Listage d'assemblage

```
                org  65021
0000   vect:    defw 0                    ;2 bytes for vector
DDE5   inter:   push ix
E5              push hl
C5              push bc
D5              push de
F5              push af
C334FE          jp   start                ;jp to music routine
F1     enprg:   pop  af
D1              pop  de
C1              pop  bc
E1              pop  hl
DDE1            pop  ix
C33800          jp   56                   ;jp to ROM ISR
21FFFF on:      ld   hl,inter             ;set up vector
22FDFD          ld   (vect),hl
F3              di
3EFD            ld   a,253
ED47            ld   i,a                  ;get vect hi-byte in I
ED5E            im   2                    ;set int mode 2
2A80FE initi:   ld   hl,(datad)           ;point to data
227EFE          ld   (locat),hl
3E01            ld   a,1                  ;switch = 1 = play note
3282FE          ld   (swtch),a
3D              dec  a                    ;delay = 0
3283FE          ld   (delay),a
FB              ei
C9              ret                       ;back to BASIC
F3     off:     di
ED56            im   1                    ;set int mode 1
FB              ei
                ret
C9     ;
3A82FE start:   ld   a,(swtch)            ;get switch in a
FE00            cp   0
CA08FE          jp   z,enprg              ;if swtch 0 then end
3A83FE          ld   a,(delay)
FE00            cp   0
CA4BFE          jp   z,nwnot              ;if delay 0 then play
3D              dec  a                    ;delay=delay-1
3283FE          ld   (delay),a            ;store new delay value
C308FE          jp   enprg                ;goto end routine
2A7EFE nwnot:   ld   hl,(locat)           ;get data location
7E              ld   a,(hl)               ;get next delay in a
FEFF            cp   255                  ;255 = data finished
CA6CFE          jp   z,reset              ;data ended so reset
3283FE          ld   (delay),a            ;store new delay
23              inc  hl                   ;point to next databyte
7E              ld   a,(hl)               ;get it in a
23              inc  hl                   ;point to next databyte
5F              ld   e,a                  ;store byte1 in e
7E              ld   a,(hl)               ;store byte2 in a
23              inc  hl                   ;point to next byte
227EFE          ld   (locat),hl           ;store next-byte-address
67              ld   h,a                  ;get frequency data...
6B              ld   l,e                  ;...in HL
110400          ld   de,4                 ;note duration of 4
CDB503          call 949                  ;call ROM beep routine
F3              di                        ;ROM-BEEP did EI, so DI
C308FE          jp   enprg                ;goto end routine
2A80FE reset:   ld   hl,(datad)           ;reset various...
227EFE          ld   (locat),hl           ;...pointers
3E01            ld   a,1
3282FE          ld   (swtch),a
3D              dec  a
3283FE          ld   (delay),a
C308FE          jp   enprg                ;goto end routine
0000   locat:   defw 0                    ;reserve memory for...
0000   datad:   defw 0                    ;various pointers...
00     swtch:   defb 0
00     delay:   defb 0
```

Page manquante
(publicité)

Page manquante
(publicité)